JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Efficient Solvers for Optimal Design Problems with PDE Constraints

Dissertation

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

Angefertigt am *Institut für Numerische Mathematik*

Betreuung:

o.Univ.-Prof. Dr. Ulrich Langer
Prof. Dr. Andreas Griewank

Eingereicht von:

Dipl.-Ing. Wolfram Mühlhuber

Linz, 15. April 2002

To the memory of my friend Gerald Scheinmayr

Abstract

In optimal design we try to improve an object by modifying its shape. These optimization problems are located on the interface to partial differential equations, numerical analysis and scientific computing. This makes the solution of optimal design problems very challenging. During recent years, the importance of optimal design has been growing, especially in the commercial market. But still nowadays, changes in the design are most often based on long lasting experience, rather than optimization methods. The main specialty of optimal design problems is that they are optimization problems governed by differential equations where we consider only the case of partial differential equations. We present strategies for the numerical solution of these optimization problems, and discuss the arising problems.

For optimal design problems with only a few design parameters we consider an approach reducing the number of parameters by eliminating the state parameters. For this reduced problem we use standard optimization methods based on sequential quadratic programming. Here, the Hessian is usually approximated by quasi-Newton formulas, like the well-known BFGS-formula. That is why, we need only function and gradient evaluations of the objective and the constraints. In most cases it is very difficult to implement gradient evaluation routines for real life optimal design problems as these involve the solution of the state problem. As an alternative to hand-coded gradient routines we consider various black-box approaches, like finite differences or automatic differentiation, and analyze their pros and cons. Combining the strengths of different approaches we realize a flexible, but also efficient method by combining automatic differentiation with hand-coded gradient routines. We demonstrate the good performance using an optimal sizing problem coming from industry.

For optimal design problems with many design parameters approaches eliminating the state equation are not suitable. We introduce an all-at-once approach considering the optimal design problem in the product space of state and design parameters. This approach treats the state equation as an equality constraint during the optimization. Besides a method based on sequential quadratic programming we introduce methods based on sequential quadratic programming and iterative regularization as we use an ill-posed problem as model problem. We analyze the well-posedness of the occurring quadratic programming subproblems in a continuous and a discrete setting. For the considered model problem, the numerical approximation of the Karush-Kuhn-Tucker systems of the quadratic subproblems leads to equation systems with large, sparse, symmetric, but indefinite matrices. We consider the numerical solution of these problems using Uzawa-type methods, reduced SQP methods or simultaneous methods. A nested iteration approach additionally accelerates the proposed method. The considered examples show the good numerical performance of the proposed method.

Zusammenfassung

In vielen Bereichen des täglichen Lebens gewinnt die rationelle Nutzung von Ressourcen eine immer größere Bedeutung. Insbesondere im industriellen Umfeld ist es heute notwendig, Produkte möglichst kosteneffizient zu erzeugen, um konkurrenzfähig zu bleiben. Gleichzeitig werden aber die Produktions- und Entwicklungszyklen immer kürzer, da der Kunde nach immer besseren Produkten verlangt. Um wettbewerbsfähig bleiben und seine Stellung am Markt behaupten oder gar verbessern zu können, muß verstärkt auf rechnergestützte Produktoptimierung gesetzt werden, obwohl nachwievor viele Änderungen ausschließlich aufgrund von Erfahrungswerten durchgeführt werden.

In der vorliegenden Arbeit betrachten wir mathematische Methoden zur Produktoptimierung. Diese versuchen durch Veränderung der Form gewisser Teile eine Verbesserung des Produkts zu erreichen, wobei beispielsweise die Lebensdauer (z.B. bei tragenden Teilen) erhöht, das Gewicht reduziert, die Produktionskosten verringert oder der Energieverbrauch reduziert werden soll. Die Lösung solch komplexer Optimierungsaufgaben erfordert das Zusammenspiel mehrerer mathematischer Disziplinen, insbesondere der Optimierung, Analysis und Numerik von Differentialgleichungen, sowie des Wissenschaftlichen Rechnens.

Bei der Frage, welche Form eines Teils für den vorgesehenen Zweck am günstigsten ist, handelt es sich um ein Optimierungsproblem, dessen Zulässigkeitsbereich durch Differentialgleichungen restringiert wird, wobei es sich hierbei meist um partielle Differentialgleichungen handelt. In dieser Arbeit sollen vor allem die numerische Lösung derartiger Optimierungsprobleme und die dabei auftretenden Schwierigkeiten diskutiert werden.

Für Probleme mit nur wenigen Designparametern ist es zweckmäßig, jene Variablen, die den Zustand in der Differentialgleichung repräsentieren, aus dem Optimierungsproblem zu eliminieren. Das dadurch entstehende Optimierungsproblem hat bei weitem weniger Parameter, wodurch Standardoptimierungsmethoden auf der Basis von SQP-Methoden verwendet werden können. Diese Verfahren approximieren die Hessematrix mittels Quasi-Newton-Formeln, wie der bekannten BFGS-Formel, sodaß nur Routinen zur Auswertung und zur Gradientenberechnung von Zielfunktional und Restriktionen notwendig sind.

Für industriell relevante Probleme ist die Implementierung von Gradientenroutinen meist sehr arbeitsaufwendig, da eine Funktionsauswertung auch die Lösung der Zustandsgleichung, im hier betrachteten Fall einer partiellen Differentialgleichung, inkludiert. Als Alternativen bieten sich die Verwendung von finiten Differenzen oder auch automatisches Differenzieren an. Nach einer Analyse der Stärken und Schwächen dieser Verfahren wird gezeigt, wie man durch Kombination von automatischem Differenzieren und handkodierten Gradientenroutinen auf flexible aber auch effiziente Art und Weise Sensitivitätsinformation berechnen kann, wobei die Vorteile dieses Zugangs anhand einer industrierelevanten Dickenoptimierung illustriert werden.

Für Probleme mit einer großen Anzahl an Designparametern ist ein Zugang, der die Zustandsvariablen eliminiert, nicht geeignet. Stattdessen wird die Zustandsgleichung als Restriktion eines Optimierungsproblems im Produktraum von Zustands- und Designvariablen betrachtet. Da als Modellbeispiel ein schlechtgestelltes Problem betrachtet wird, werden nicht nur Lösungsverfahren auf Basis von SQP-Methoden, sondern auch solche, die SQP-Methoden mit Regularisierungsverfahren kombinieren, eingeführt. Es wird sowohl die Wohldefiniertheit der präsentierten Verfahren als auch deren numerische Approximation betrachtet. Letztere führt auf grossdimensionierte Gleichungssysteme mit symmetrischen, indefiniten Matrizen, welche mit Uzawa-artigen Verfahren, reduzierten SQP-Verfahren, aber auch simultan gelöst werden können. Zusätzlich wird durch Verwendung geschachtelter Ansatzräume das vorgestellte Verfahren beschleunigt.

Preface

After I had finished my diploma in September 1997, I was looking for a position in Linz because I was renovating an old house in Linz at that time. Prof. Ulrich Langer offered me a position as research assistant at a new research programme starting in March 1998. I joined the new *Special Research Initiative – Numerical and Symbolic Scientific Computing* and started to write my PhD.

Optimal design problems were not really the topic of my choice because I had only very little knowledge on that. Nevertheless, I have learned to appreciate the hidden beauties of this field up to now. During that time, Ewald Lindner spent much time explaining optimization stuff to me for which I want to thank him a lot.

In summer 1999 Prof. Andreas Griewank visited our university to give a course on Automatic Differentiation. Up to that time I mainly used finite differences to get gradient information and had focused on accelerating the function evaluations by using fast iterative solvers for the state equation. Using automatic differentiation I was able to solve also problems with a hundred design parameters, an amount I had not dared to dream of a few months before. Thanks also to Olaf Vogel and Andrea Walther.

Then, the development slowed down. I realized, that the strategy used up to that time was only able to solve optimal design problems with a few hundred design parameters. At that time, my colleague Stefan Reitzinger helped me a lot to overcome this scientific low.

At the same time, Martin Burger explained me inverse problems from a point of view which was very similar to what I did for optimal design problems. Getting motivated by his explanations we developed an all-at-once strategy for inverse problems which could also be used for optimal design problems, although this leads to additional difficulties. Thank you for the good collaboration during that time.

During all the years both supervisors, Prof. Langer and Prof. Griewank guided my work and I am very thankful for all the discussions which led to several improvements of my work. I also want to thank all the other colleagues at the *Special Research Initiative* and at the *Department for Computational Mathematics* for their interest in my work and for the good working climate. I do not want to forget my parents, my brother and Anke, for all their support through the years. Thanks to all of them.

Last but not least, this work would not have been possible without the financial support of the Austrian Science Fund *Fonds zur Förderung der wissenschaftlichen Forschung (FWF)* within the subproject F1309 of the Special Research Initiative (SFB) F013 – *Numerical and Symbolic Scientific Computing*.

Notations and Abbriviations

$\mathbb{R}$, $\mathbb{R}^d$	–	Set of real numbers and set of vectors $\mathbf{x} = (x_i)^T_{i=1,..,d}$, $x_i \in \mathbb{R}$, $i = 1,..,d$.
d	–	Space dimension.
u	–	Scalar or vector valued function.
u_h, $\mathbf{u}$	–	Finite element function and its vector representation.
$\mathbf{K}$	–	System matrix $\mathbf{K} \in \mathbb{R}^{m\times m}$.
$\langle\cdot,\cdot\rangle$	–	Duality product or scalar product in an Hilbert space.
$\|\cdot\|_Z$	–	Norm in Z.
Ω, $\Gamma = \partial\Omega$	–	Bounded domain (open and connected subset of $\mathbb{R}^d$, $d = 1, 2, 3$) with sufficiently smooth boundary $\Gamma = \partial\Omega$.
$\mathbf{n}$	–	Normal unit (outward) direction with respect to the boundary $\Gamma = \partial\Omega$ of some domain Ω.
grad	–	Gradient, $\operatorname{grad} u(x) = \left(\frac{\partial u(x)}{\partial x_i}\right)^T_{i=1,..,d}$ for $x \in \mathbb{R}^d$.
grad_λ, ∇_λ	–	Gradient with respect to λ.
$\nabla^2 J$	–	Hessian of J.
Δ	–	Laplace operator, $\Delta u(x) = \sum_{i=1}^d \frac{\partial^2 u(x)}{\partial x_i^2}$ for $x \in \mathbb{R}^d$.
$\mathbf{I}$	–	Identity matrix.
E	–	Observation operator.
$\mathbf{A}^T$	–	Transpose of $\mathbf{A}$.
$\dim(\mathbf{u})$	–	Dimension of the vector $\mathbf{u}$.
meas Γ	–	Measure of the set Γ.

δ_{ij}	–	Kronecker's delta, $\delta_{ij} = 1$ for $i = j$, $\delta_{ij} = 0$ for $i \neq j$.
$L^2(\Omega)$	–	Space of scalar square–integrable functions on Ω.
$(L^2(\Omega))^d$	–	Space of vector valued square–integrable functions on Ω.
$L^p(\Omega)$	–	$L^p(\Omega) = \{f : \Omega \to \mathbb{R} \mid u$ is Lebesque measureable, $\|u\|_p < \infty\}$ with $\|u\|_p = \left(\int_\Omega |u|^p \,\mathrm{d}x\right)^{\frac{1}{p}}$.
$H^1(\Omega)$	–	$H^1(\Omega) = \left\{v \in L^2(\Omega) \mid \operatorname{grad} v \in (L^2(\Omega))^d\right\}$.
$H^{1/2}(\partial\Omega)$	–	Trace space of $H^1(\Omega)$.
$H_0^1(\Omega)$	–	$H_0^1(\Omega) = \left\{v \in H^1(\Omega) \mid v = 0 \text{ on } \partial\Omega\right\}$.
$(H^1(\Omega))^d$	–	$(H^1(\Omega))^d = \left\{\mathbf{v} \in (L^2(\Omega))^d \mid \frac{\partial v_i}{\partial x_j} \in L^2(\Omega)\ \forall i, j = 1, \ldots, d\right\}$.
U	–	State space.
U_h	–	Discretized state space.
Q	–	Design space.
Q_h	–	Discretized design space.
Z	–	Space for the data z.
Y	–	Image space of the state equation e.
U^*	–	Dual space of U.
u	–	State variable.
q	–	Design variable.
λ	–	Lagrangian multiplier.
u_h, q_h	–	Discretized state and design variable.
$\mathbf{u}, \mathbf{q}$	–	Vector representation of discretized state and design variable.
z, z^δ	–	Exact and noisy data.
J	–	Objective.
$e(u, q)$	–	State equation.
$\mathcal{L}$	–	Lagrangian.
m	–	Number of state parameters.
n	–	Number of design parameters.
$\mathbf{u} = (u_1, \ldots, u_m)$	–	Components of a vector.
AD	–	Automatic differentiation.
AMG	–	Algebraic multigrid.
CAD	–	Computer aided design.

CG	–	Conjugate gradient.
FE	–	Finite element.
FEM	–	Finite element method.
GMRES	–	Generalized minimal residual.
IRSQP-method	–	Iteratively regularized SQP method.
KKT-system	–	Karush-Kuhn-Tucker system.
LM method	–	Feasible path Levenberg-Marquardt method.
LMSQP method	–	Levenberg Marquardt SQP method.
MINRES	–	Minimal residual.
MG	–	Multigrid.
MMA	–	Method of moving asymptotes.
PDE	–	Partial differential equation.
PSSQP method	–	Product space SQP method.
QMR	–	Quasi minimal residual.
QP	–	Quadratic programming.
SQP	–	Sequential quadratic programming.

Contents

Chapter 1

An Overview to Optimal Design Problems

1.1 Introduction

Optimization plays a role of increasing importance in today's every day life. Its basic principle is very simple: Usually one has a set of parameters which describe e.g. a form, a path, quantities to buy or sell or a capacity to store certain goods. Each element of this set of parameters is rated by a so-called cost functional or objective. The goal of an optimization is to find a parameter vector to minimize the cost. Typical costs are as the name implies prices or costs in the financial sense, but may also be the drag of an airplane, the loss of energy, the time to follow a path, the length of a journey or the weight of a structure, to name only a few, or a (non) linear combination of (some of) them. Usually, one has to fulfill additional constraints during the optimization which restrict the choice of the parameter.

Optimal design problems are no typical optimization problems, although they fulfill the requirements above. These problems are located on the interface of different fields, and only one of them is optimization. The others are

- partial differential equations,
- numerical analysis,
- scientific computing, and last but not least, for the numerical realization,
- information technology.

This peculiarity makes the solution of optimal design problems rather complicated but on the other hand very interesting.

In optimal design one tries to improve an object by modifying its shape. The quality is measured by a criterion which can be interpreted as a cost-functional in the optimization sense. During recent years, the importance of optimal design has been growing, especially in the commercial market. But still nowadays, changes in the design are most often based on long lasting experience, rather than optimization methods. The main specialty of optimal design problems is that they are governed by differential equations, in many cases partial differential equations (PDEs) or even systems of coupled PDEs.

PDEs describe many physical models, e.g.

- the *Maxwell equations* electromagnetic fields (see e.g. IDA AND BASTOS [91] and references therein, or KOST [100]),
- the *Cauchy-Navier equations* model continuum mechanics (see e.g. CIARLET [40] or HUGHES [90]), or
- the *Navier-Stokes equations* describe the dynamics of fluids (see e.g. GIRAULT AND RAVIART [64]).

Usually, analytical solutions for PDEs are not known. That is why, numerical approximation schemes are used for the calculation of approximate solutions. Nowadays, the most popular discretization technique is certainly the finite element method, which is based on a variational formulation of the PDE. For linear PDEs the finite element method leads to a large, sparse linear equation system which has to be solved. For time dependent problems appropriate time-integration schemes have to be used, for non-linear problems Newton's method or a fix-point approach.

One key-task in the approximate solution of a PDE is the solution of a large sparse linear equation system. For non-linear or time-dependent problems such a kind of system has to be solved repeatedly, e.g. once per time step or non-linear iteration step.

As long as the number of unknowns in this linear equation system is not too large, direct solution methods, e.g. variants of Gaussian elimination method (see e.g. GEORGE AND LIU [57] or DUFF, ERISMAN, AND REID [47]) can be used. Software packages implementing these techniques are e.g. SPARSEKIT [131] of SUPERLU [45]. These methods try to reduce the costs by reordering rows and columns in such a way that the fill-in is minimized, i.e. they try to minimize the number of nonzero elements introduced by the factorization process at positions where the original matrix was zero. For large systems of equations, these direct elimination methods become inefficient and iterative methods have to be used.

Iterative methods exploit sparsity to a much higher extent than direct elimination methods. They mainly need matrix-vector multiplications and other operations whose calculation time is proportional to the number of unknowns. The best-known iterative methods are Krylov subspace correction methods (see e.g. HACKBUSCH [77], AXELSSON [3], MEURANT [112] or SAAD [132]). Examples for well-known Krylov subspace correction methods are

- the *Conjugate Gradient Method* by HESTENES AND STIEFEL [85],
- the *Minimal Residual Method* by PAIGE AND SAUNDERS [117],
- the *Quasi-minimal Residual Method* by FREUND AND NACHTIGAL [56], or
- the *Generalized Minimal Residual Method* by SAAD AND SCHULTZ [133].

Appropriate preconditioners are necessary for fast convergence of these methods. Therewith it is possible to construct solvers of optimal order, i.e. the CPU-time and the memory requirements are proportional to the number of unknowns. Multigrid methods, respectively multigrid preconditioners fulfill these requirements (JUNG AND LANGER [92]). They are based on a well-balanced interplay between a smoothing operator and a coarse grid correction. One possibility to calculate the coarse grid correction is by using nested finite element spaces. This is called *geometric multigrid method*. If no nested finite element spaces are available, *algebraic multigrid methods* can be used which construct a matrix hierarchy only by using fine-grid information.

Coming back to optimal design problems in general, we want to emphasize that these are PDE-constrained optimization problems, although this term contains by far more. In the following sections we want to introduce three classes of optimal design problems, namely

- optimal sizing problems,
- boundary shape optimization problems, and
- topology optimization problems.

These problems have in common, that the design parameters influence the domain in which the PDE has to be fulfilled in some sense. Two other problems classes which are not treated here, are of similar structure, namely

- inverse problems, and
- optimal control problems.

We will make a few general remarks on inverse problem, as one of our model examples is an inverse problem. Optimal control problems are not treated here, for these we want to refer to the literature. At the end of this chapter, we introduce an abstract problem class which will be treated in the following chapters.

1.2 Optimal sizing problems

Optimal sizing problems are PDE constrained optimization problems where the design parameters influence the domain in which the PDE has to be fulfilled. This is also valid for boundary shape optimization and topology optimization. The main specialty of optimal sizing problems is the easy dependence of the geometry on the design parameters.

Optimal sizing is a $2\frac{1}{2}$-dimensional optimization, where the design parameter is the thickness over a constant cross section, i.e. the parameter dependent domain is of the form

$$\Omega(q) = \big\{(x_1,x_2,x_3) \in \mathbb{R}^3 \mid (x_1,x_2) \in \omega, x_3 \in (-q(x_1,x_2), q(x_1,x_2))\big\} \tag{1.1}$$

where ω denotes the cross section and $0 < \underline{q} \le q(x,y) \le \overline{q}$, $\underline{q}, \overline{q} \in \mathbb{R}^+$. In the design domain $\Omega(q)$ the Lame-equations have to be fulfilled, i.e.

$$\hat{a}(q;u,v) = \hat{F}(q;v), \qquad \forall\, v \in \hat{U}(q) \tag{1.2}$$

with

$$\hat{a}(q;u,v) = \int\limits_{\Omega(q)} \frac{\partial u_i}{\partial x_j} E_{ijkl} \frac{\partial v_k}{\partial x_l}\,\mathrm{d}x, \quad \hat{F}(q;v) = \int_{\Omega(q)} \langle f,v\rangle\,\mathrm{d}x + \int_{\Gamma_N(q)} \langle g,v\rangle\,\mathrm{d}s$$

where E_{ijkl} denotes the elasticity tensor, f the volume force density and g the surface force density on a part Γ_N of the boundary. We use Einstein's summation convention where necessary. $\hat{U}$ denotes the set of admissible displacements, i.e.

$$\hat{U}(q) = \big\{v \in [H^1(\Omega(q))]^3 \mid v = 0 \text{ on } \Gamma_D, \text{meas } \Gamma_D > 0\big\} \tag{1.3}$$

where $\partial\Omega = \Gamma_D \cup \Gamma_N$, $\Gamma_D = \overline{\Gamma}_D$ and $\Gamma_D \cap \Gamma_N = \emptyset$.

Typical problem settings are e.g.

- minimize the mass

$$\int\limits_{\Omega(q)} \rho(x)\,\mathrm{d}x$$

  where ρ denotes the density of the material, under constraints on the displacements or on the maximal stress, or

- minimize the variance of the stress under constraints on the maximal stress; this shall result in an equal-distribution of the stresses.

Up to now, this is a very general optimal design problem. For optimal sizing we additionally assume:

- We consider a plane stress problem, i.e. $\Omega(q)$ is thin in x_3 direction.
- Ω is considered as a plate that can carry only stresses parallel to the $x_1 - x_2$ plane.
- The applied surface tractions g and the body forces f are independent of x_3. Additionally Γ_N does not depend on q.
- Last but not least, we want to assume that no displacements in x_3-direction exist and that the displacements in x_1 and x_2-direction are both independent of x_3.

Under these assumptions we can simplify the problem (1.2), which leads to

$$a(q;u,v) = F(q;v), \qquad \forall\, v \in U,$$

with

$$a(q;u,v) = \int\limits_{\omega} q\,\frac{\partial u_i}{\partial x_j}\,E_{ijkl}\,\frac{\partial v_k}{\partial x_l}\,\mathrm{d}x$$

and

$$U = \left\{v \in [H^1(\omega)]^2 \mid v = 0 \text{ on } \gamma_D, \text{meas } \gamma_D > 0\right\} \tag{1.4}$$

In the integral defining the bilinearform the summation runs only from one to two, as the domain ω is twodimensional. The main advantage of this formulation is that the parameter does no more influence the computational domain, but is a scalar multiplier in the state equation. E_{ijkl} denotes the material elasticity tensor, whereas $q \cdot E_{ijkl}$ can be interpreted as the effective elasticity tensor. Due to the assumptions on the external forces, $\hat{F}(q;v)$ can also be rewritten in a form that contains only integrals over constant domains, i.e.

$$F(q;v) = \int_{\omega} q\,\langle f,v\rangle\,\mathrm{d}x + \int_{\gamma_N} \langle g,v\rangle\,\mathrm{d}s$$

where γ_D and γ_N denote the boundary part of ω corresponding to Γ_D and Γ_N, respectively. As can be seen, the design parameter q appears only as a scalar multiplier in the variational formulation. In most cases, q is discretized by a piecewise constant function when using the finite element method. Then, the thickness plays the role of a multiplier of the element stiffness matrix which simplifies the calculation of derivatives of the state equation a lot.

This problem class was treated by several people. The first to solve this problem numerically were Rossow and Taylor [128] in 1973 using the finite element method.

MAHMOUD [111] used this approach for minimizing the weight of a unit injector rocker arm. For a faster evaluation of the gradients he used approximation models to approximate the objective and the constraints in a trust region of the current iterate. For a class of nonlinear materials STANGL [142] showed existence of a solution for such a problem using fixed-point arguments.

In Chapter 4 we will use an optimal sizing problem as a model example for an optimal design problem with a small number of design parameters. We will focus our interest mainly on the numerical solution and the realization of such a method in an industrial environment. Most attention will be paid to a fast and flexible gradient evaluation and to the possibility to handle very complicated objectives and constraints.

This problem was also considered from a completely different point of view. By allowing the design variable to take values close to zero, this problem changes its nature to a topology optimization problem. Then, the maximization of the stiffness which is equivalent to the minimization of the compliance is considered as objective. For this problem CÉA AND MALANOWSKI [35] showed existence of a solution for strictly positive lower thickness bound. PETERSSON [118] generalized the results to a problem with zero lower thickness bound and unilateral displacement constraints. Convergence of a finite element analysis in L^p was shown again by PETERSSON [119] under certain assumptions on the optimal stress field. For publications dealing with the problem of checkerboard patterns see Section 1.4.

1.3 Boundary shape optimization problems

Boundary shape optimization problems are in some sense a generalization of optimal sizing problems. Optimal sizing problems are a $2\,\frac{1}{2}$-dimensional optimization with a very simple parameterization of the domain. Additionally, various assumptions were made to reformulate the state problem on a fixed domain. In many problems these assumptions can not be fulfilled. Furthermore, the domain can not be represented in the form needed for an optimal sizing problem.

In boundary shape optimization the design parameter is the boundary of the computational domain. In this sense it is a generalization of an optimal sizing problem. As the dependency of the computational domain on the corresponding parameter can be very general we can not assume to be able to transform the state problem into a state problem on a fixed domain. On the other hand, this enables us to handle a much larger problem class.

One of the most difficult parts in the numerical treatment of boundary shape optimization problems is the parameterization of the boundary. The main problem is that the results of the subsequent design optimization depend on the used parameterization in a critical way, as the parameterization determines the set of admissible designs. That is why, the choice of the parameterization also restricts the possible gain in the objective.

For illustration, let us consider the following example: Assume you have a boundary shape optimization in 2D. The whole boundary is kept fixed, except one part between two points. If one uses a straight line for parameterizing this boundary, the set of admissible domains has only one point, as the two endpoints determine all parameters of the line. When using a circular arc, one design parameter is left (describing e.g. the radius of the circular arc). Using a b-spline curve of a fixed order gives us more degrees of freedom, but still the admissible set of designs is limited by the choice of the parameters of the b-spline (e.g. the number of control nodes or the degree).

In practice, two different approaches are used:

- Either the domain is parameterized using geometric parameters, or
- for the calculation, a discretization of the boundary is used as parameterization.

The use of geometric parameters is of course coupled to using geometric primitives to describe the boundary. Geometric primitives have a natural parameterization, e.g. a circle its center and radius. Additionally, constraints have to ensure the consistency of the geometry, i.e. the end of one edge is the starting point of another, that the boundary shall not self-intersect, or that the connection of two edges should be smooth. All these constraints have to be incorporated into a geometry handler mapping the vector of design parameters to an admissible geometry, which makes the implementation rather challenging. In 2D this can still be done, as the boundary consists only of curves, but in 3D this can be very complicated. Commercial computer aided design (CAD) tools often have the functionality to maintain a parametric model of a geometry. But usually it is very complicated to integrate these tools into an optimal design code, as the geometry handler has to provide derivative information, at least if gradient based optimization routines are used. Additionally, constraints on the geometry may exist, which can not be easily represented in the parameter domain, e.g. the minimal distance between two opposite edges is limited from below.

Often, one tries to escape some of the problems described above by using spline-curves and spline-surfaces to represent the design boundary. Then, some of the parameters of the spline are the design parameters. Usually, the location of the control knots and their weight (if rational b-splines are used) is taken as design parameter and the degree remains fixed. For details on the representation of splines see e.g. HOSCHEK AND LASSER [89]. EGARTNER AND SCHULZ [48] use this approach in the design of turbine blades. Also in aerodynamic optimization, e.g. airfoil design, it is often applied (e.g. BARTELHEIMER [6]).

The main advantage of this description is that the number of design parameters is usually rather low, especially in 2D. Nevertheless, problems with several hundred design parameters are often found in 3D. But in most cases, it is still possible to apply optimization routines based on dense linear algebra, like NLPQL by SCHITTKOWSKI [137] or NPSOL by GILL, MURRAY, SAUNDERS, AND WRIGHT [62].

When using a discretization of the boundary also for parameterizing the boundary, each boundary node of the FE mesh is one design parameter. This approach does not need a complicated geometry handler based on a CAD system. That is why, this approach is often named *CAD-free parameterization* (see e.g. MOHAMMADI AND PIRONNEAU [113]). This approach is very flexible, but usually results in problems with very many design parameters. Additionally, one has to restrict the set of shapes which can be described by this approach to a set of desired shapes. This can be done e.g. by introducing bounds on the maximal curvature of the boundary (see e.g. LUKÁŠ [108]) or by using smoothing operators as proposed by MOHAMMADI AND PRIONNEAU [113].

When doing boundary shape optimization it is necessary to avoid re-meshing of the computational domain during one optimization step, as re-meshing usually introduces jumps in the objective. That is why not only a parameterized geometry but also a parameterized mesh is needed. An often used alternative are mesh-moving strategies. These try to deform the mesh of a reference geometry to get a mesh of the current one. In order to get good results, tests on the quality of the computational mesh which are also used in mesh generation have to be included in these mesh-moving strategies.

The use of mesh-moving algorithms has usually also implications on the optimizer. As these algorithms are not very stable with respect to very large deformations it is better to use trust region methods. By the trust region parameter it is rather easy to restrict the maximal change of the design parameters and therefore of the geometry. When using line search methods more robust mesh-moving algorithms are needed.

Most of the aspects presented above deal with the realization of a boundary shape optimization by discretizing the problem and then optimizing this discretized optimization problem. Nevertheless, derivatives of the objective with respect to the boundary can also be calculated by a completely different approach, so-called *shape-derivatives*. Sokolowski and Zolésio [141] and Delfour and Zolésio [44] focus on this approach. A shape-derivative is a derivative of the continuous objective with respect to the boundary. Usually this results in a partial differential equation for the gradient. For numerical purposes this PDE has to be discretized, but the solution of this discrete problem is then not a gradient of the discretized objective any more. That is why, for numerical purposes the previously described concept is often preferred. In contrast, shape derivatives are often used to show existence or uniqueness of solutions (see e.g. Dambrine and Pierre [43]).

Besides these approaches, also other ones exist, e.g. using ficticious domains (Kunisch and Peichl [103]) or using level set methods (Hintermüller and Ring [86]). But up to now these methods were not used for real life problems.

1.4 Topology optimization problems

As explained in Section 1.3 boundary shape optimization problems depend strongly on the way the design boundary is parameterized. Additionally, the results depend strongly on the topology of the design domain, as boundary shape optimization can not change the topology e.g. by adding or removing holes.

Topology optimization problems are material distribution problems. For these problems the density of the material is the design parameter. In each point of the computational domain, the density should be either one or zero, indicating material or void respectively. In some sense, the structure of these problems is quite similar to optimal sizing problems, where the thickness played a similar role.

The best analyzed topology optimization problem is the maximum-stiffness problem, often called also minimal-compliance problem which looks as follows: Find a density $q \in Q$ and a state $u \in U$, such that

$$\begin{aligned} & F(u) \to \min_{(u,q)\in U\times Q} \\ \text{subject to} \quad & a(q;u,v) = F(v), \qquad \forall\, v \in U, \\ & \int_\Omega q \,\mathrm{d}x = V_0, \\ & 0 < \underline{q} \le q \le 1 \end{aligned} \tag{1.5}$$

with

$$a(q;u,v) = \int_\Omega \rho(q)\,\frac{\partial u_i}{\partial x_j}\,E_{ijkl}\,\frac{\partial v_k}{\partial x_l}\,\mathrm{d}x$$

and

$$F(v) = \int_\Omega \langle f, v \rangle \, \mathrm{d}x + \int_{\gamma_N} g\, v \, \mathrm{d}s.$$

V_0 denotes a prescribed volume, f the volume force density, g prescribed surface tractions on Γ_N. For defining the bilinearform Einstein's summation convention is used. We introduced the parameter q to be in $[\underline{q}, 1]$ with $\underline{q} > 0$. In order to prevent q to attain intermediate values, special functions $\rho(q)$ are used, e.g.

$$\rho(q) = q^m \tag{1.6}$$

for a given m which is known as SIMP (Solid Isotropic Material with Penalization, see e.g. BENDSØE [12]) or

$$\rho(q) = \frac{q}{1 + (m-1)(1-q)}$$

which was introduced and analyzed by STOLPE AND SVANBERG [144]. m is usually chosen larger than 1. BORRVALL AND PETERSSON [24] presented an alternative to prevent q to attain intermediate values. They allow m to be 1 in (1.6), but introduced an additional constraint. An overview on many aspects of topology optimization is presented by the monograph of BENDSØE [13] and the review articles by ESCHAUER AND OLHOFF [53] or ROZVANY [129].

The main difficulty is that the problem (1.5) is ill-posed, i.e. there are generally no solutions. BORRVALL AND PETERSSON [24] motivated that the set of feasible designs is not *sufficiently closed*, i.e. it is not closed with respect to the relevant topology. This introduces several effects which are quite typical for problem (1.5). On the one hand, mesh dependent solutions can appear. These are a consequence of the fact that a solution of the continuous problem need not exist. On the other hand, checkerboard patterns and other numerical anomalies occur (see e.g. BENDSØE [13]), which are a consequence of non-convergence when the mesh is refined. All solution methods now try to enlarge or restrict the set of admissible designs in such a way that the new set is closed (with respect to a suitable topology) and solutions exist.

One possibility is to restrict the gradient of the design parameter, i.e.

$$\| \operatorname{grad} q \| \leq c \tag{1.7}$$

where the norm is taken in some L^p space. Depending on the choice of p, we distinguish several problems:

For $p = 1$, (1.7) can be interpreted as a bound on the total variation of q. This restriction is usually called *perimeter constraint* and was first numerically treated by HABER, JOG, AND BENDSØE [75]. PETERSSON [120] provides results on the existence of solutions, as well as on convergence aspects. Unfortunately, the advantage of perimeter constraints seems mainly of theoretical nature. From the numerical point of view, it seems that a solution algorithm is in general unstable and sensitive to local optima (HABER, JOG, AND BENDSØE [75]).

A different approach are *slope constraints* which correspond to (1.7) for $p = \infty$. These were introduced in the optimal design of elastic plates by NIORDSON [114]. In topology optimization they were first used by PETERSSON AND SIGMUND [121] where bounds on the directional derivatives were used. Again, existence of a solution can be shown. From the numerical point of view, this approach results in many constraints. PETERSSON AND SIGMUND [121] reported this as a significant drawback of the method. ZHOU, SHYY, AND THOMAS [161] presented an algorithm which exploits the characteristics of the constraints. They reported that the incorporation of the constraints required hardly any extra computational cost.

The use of L^p norms for $1 < p < \infty$ was never treated numerically in literature.

A completely different approach are restrictions using filters. By introducing a low-pass filter, the unwanted high frequency components in the design variable can be reduced. Several approaches were proposed in literature:

SIGMUND [140] used a filter for the sensitivities of the objective in order to prevent too thin structures. Although there is no theoretical justification at the moment, this approach proved to be quite effective. Moreover, the solutions seem to be mesh independent.

BRUNS AND TORTORELLI [32] apply a mollifier between every optimization iterate to stabilize the numerical procedure and to filter high frequency components of q. BOURDIN [25] showed the existence of solutions for this approach and also the influence of the filter radius on the solution.

Usually, in (1.6) m is taken larger than 1. In BORRVALL AND PETERSSON [24] a filter method is presented which works also for $m = 1$. In order to eliminate intermediate densities, they penalize them. To get also existence of solutions, they use a mollifier to smooth the density. In BORRVALL AND PETERSSON [23] results for large scale optimization problems in 3D are presented using this approach.

The main advantage of filter methods to methods bounding the gradient is that it is possible to prove mesh independence of the numerical solution (see BOURDIN [25] and BORRVALL AND PETERSSON [24] for finite element convergence of the latter two methods). For a good and detailed overview of these methods see BORRVALL [22].

For the numerical solution most papers exploit the special structure of problem (1.5) to derive an efficient solver. SVANBERG [145] developed the *Method of moving asymptotes (MMA)* which is still nowadays used by most topology optimization codes. It is based on a convex and separable expansion of the objective, which accelerates the calculation of a search direction in the optimization.

An exception is the paper of MAAR AND SCHULZ [110]. They use an interior point method for optimizing the problem. Each quadratic programming problem is solved by a multigrid method using transforming smoothers. As they used no regularization, they got mesh-dependent results which could also be seen at the presented pictures. Nevertheless the approach itself seems interesting, although the code is not faster than codes based on MMA up to now.

As mentioned above, most topology optimization papers focus on a solution of (1.5). Recently, HOPPE, PETROVA, AND SCHULZ [87, 88] presented a solution procedure for a topology optimization in electro-magnetics. Up to now, no analysis is available, nevertheless the numerical results look very promising.

1.5 Inverse problems

Inverse problems are concerned with determining causes for desired or observed effects. Therefore, they appear quite frequently. A very important subclass are *parameter identification problems* (see e.g. BANKS AND KUNISCH [5] or OMATU AND SEINFELD [116]). There, distributed parameters in an underlying model (usually a partial differential equation) are determined from indirect measurements. We will use a member of this class as model example in Chapter 5.

The majority of inverse problems is *ill-posed*, i.e. either the solution does not exist in a strict sense, or solutions might not be unique and / or might not depend continuously on the data. Therefore, regularization methods have to be used in order to obtain a stable approximation of the solution in the presence of data noise (introduced e.g. by a measuring device). We refer to Engl, Hanke, and Neubauer [50] and to Kirsch [99] for an overview of regularization methods for inverse ill-posed problems.

The most well-known classical approach to regularize an inverse problem is *Tikhonov regularization* (see e.g. Chavent and Kunisch [37] or Engl, Kunisch, and Neubauer [51]). There we replace the least-squares problem by a close stable problem. Recently, also the application of *iterative regularization methods* became more and more popular (see e.g. Hanke, Neubauer, and Scherzer [80], Hanke [79], or Kaltenbacher [94, 95]). The regularizing effect of an iterative regularization method comes form the early termination of the iteration procedure, where the stopping index is chosen in dependence of the noise level. We refer to the survey paper by Engl and Scherzer [52] for an overview.

1.6 Abstract problem class

All previously presented problems have in common that each of these problems can be described as a PDE constrained optimization problem. Of coarse, the objective has certain specialties depending on the specific type of problem which can be exploited in numerical algorithms. Additionally, due to the similar nature, ideas from one problem type can be transferred to another.

In general, in each of these problems we try to solve an optimization problem

$$J(u,q) \to \min_{(u,q)\in U\times Q} \tag{1.8}$$

under a constraining state equation

$$e(u,q) = F. \tag{1.9}$$

Additionally, other constraints on the parameter $q \in Q$, the state $u \in U$ or on both variables may be present.

The objective J can be of different type: For the presented topology optimization problem it is a linear function in u and does not depend on q explicitly. For inverse problems it is usually a function of least-squares type comparing the observation which depends on the state solution with the given data. Additionally, it contains usually a regularization term. For sizing and boundary shape optimization often very general objectives can be found, in many cases depending on the state solution u alone. Then, the parameter appears only in the constraints. Especially for these rather general functions it is often difficult and very time consuming to calculate gradient information which is needed by the optimizer. In these situations, automatic differentiation (AD) can help a lot. The main references of this technique are the proceedings of the AD conferences in Breckenridge [68], Santa Fe [15] and Nice [42], and the monograph by Griewank [67]. We will show how this technique can be used to accelerate the development of routines for the calculation of derivatives.

The state equation can also be of very different type. We will treat only problems of elliptic type here, nevertheless optimization problems constrained by time-dependent or nonlinear state equations are of high practical importance. We want to mention also the field of *multidisciplinary design optimization* which is gaining more and more attention during the last years. Here, the state parameter consists of several state variables describing different physical quantities (e.g. mechanical displacements of an airfoil and the surrounding flow field). The constraining state equation is in most cases a coupled field problem (e.g. a fluid-solid interaction). For this problem class already the solution of the direct problem is very challenging (for examples coupling electric and mechanical fields see e.g. ALURU AND WHITE [2], KALTENBACHER, LANDES, LERCH, AND LINDINGER [96], KALTENBACHER, LANDES, NIEDERER, AND LERCH [97] LERCH, KALTENBACHER, LANDES, AND LINDINGER [106], or WACHUTKA [155]).

The remaining part of this work is organized as follows: In Chapter 2 we introduce the basic ingredients which will be necessary in the remaining chapters. We give a short overview on constrained optimization and the numerical solution of elliptic partial differential equations using the finite element method. Additionally, we give an introduction into the numerical solution of large sparse linear equations using direct elimination methods or iterative methods with appropriate preconditioning.

Chapter 3 contains a short introduction into automatic differentiation. After motivating its basic ideas, we present the two basic calculation strategies for derivatives – directional derivatives and gradients – in the concept of automatic differentiation. Although the basic principle is the chain rule which is known from basic calculus, this presentation is necessary to understand the properties of the forward and the reverse mode, as well as their drawbacks. This section is completed by a short presentation of two classes of AD tools, tools based on source to source transformation and tools based on operator overloading.

In Chapter 4 we present a new method for solving optimal design problems with few design parameters. At the beginning we introduce a model problem which originates from an industrial design process. This problem will be used in the following considerations. The main part is the investigation of different gradient calculation strategies and an analysis of their pros and cons. We end up with a new method for the efficient calculation of derivatives which combines automatic differentiation with hand-coded gradient routines. Numerical results show the efficiency of this method.

Chapter 5 starts with an analysis of the properties of the optimization strategies presented in Chapter 4. We work out, why this strategy can only be used for optimal design problems with few design parameters. After introducing a model problem coming from parameter identification, we will introduce and analyze an optimization method working in the product space of design and state space. Unlike the approach in Chapter 4 this approach treats the state equation as a constraint during the optimization and does not formally eliminate it. This introduces additional difficulties but enables us to solve also design problems with large design spaces. The numerical solution is based on a sequential quadratic programming method, where we use iterative methods for solving the underlying QP problem. Hierarchical strategies can be used for gaining additional speedup. Numerical results showing the efficiency conclude this section.

In Chapter 6 we present a software design for the implementation of optimal design problems. It is based on the strict splitting of the optimizer and the optimization problem. The optimization problem itself is subdivided into three parts: One establishing the communication between optimizer and state problem solver, into the state problem itself and into the objective. The latter is treated as a function mapping the product space $U \times Q$ into the real numbers where U denotes the state space and Q the design space. These three parts are described in more detail, as well as the functionality they have to provide to realize the solution strategies for optimal design problems presented here.

Chapter 2

Basic Ingredients

2.1 Introduction

For the numerical treatment of optimal design problems as well as inverse problems various basic ingredients are necessary which originate from different fields. On the one hand it is quite natural that the area of optimization is of major importance. But due to the special structure of the problem (1.8), (1.9) it is necessary to have also a good overview on the numerical solution of partial differential equations. This is a very wide field, where many different aspects need to be treated. That is why we want to limit ourselves here to elliptic problems and their discretization using the finite element method. Optimal design problems for parabolic and hyperbolic problems are also of high practical importance, but the numerical treatment of the forward problem is already completely different and they are therefore excluded (cf. e.g. Quarteroni and Valli [123] or Grossmann and Roos [72]). The usual solution method for elliptic problems using the finite element method is as follows:

- Subdivide the computational domain into geometric elements on which finite elements are defined. These form a finite dimensional test and solution space for the variational form of the partial differential equation.
- By replacing the continuous test and solution spaces by their finite dimensional approximations the variational form of the partial differential equation is equivalent to a set of equations, possibly non-linear.
- The central part of most numerical solution schemes for this set of equations is a solver for a set of linear equations. As the coefficient matrix is large and sparse, nowadays more and more iterative solvers using appropriate preconditioners are applied. These replace the up to now used direct solvers, especially for systems with a very large number of unknowns.

This chapter shall give an overview on the various tools needed in the later chapters. We give a short introduction into constrained optimization, as well as into the finite element method. Last but not least we present an overview on some iterative solution methods for linear systems of equations and to remember some basic facts on preconditioning, especially to multi-grid preconditioners.

2.2 Constrained optimization

This section deals with the optimization of a given objective where several constraints on the variables have to be enforced. For simplicity, we want to restrict ourselves to a finite dimensional setting. A general formulation of this problem is

$$\begin{aligned} & J(\mathbf{x}) \to \min_{\mathbf{x}\in\mathbb{R}^n} & & \\ \text{subject to} \quad & c_i(\mathbf{x}) = 0, & & i \in I_1, \\ & c_i(\mathbf{x}) \le 0, & & i \in I_2. \end{aligned} \tag{2.1}$$

All the functions J and $c_i, i \in I_1 \cup I_2$, are assumed to be sufficiently smooth and real-valued, I_1 and I_2 are finite index sets. J denotes the objective, $c_i, i \in I_1$, the equality constraints, $c_i, i \in I_2$, the inequality constraints. In unconstrained optimization one can specify conditions on J which are necessary or even sufficient for the existence of a local minimum. In constrained optimization these conditions do not operate on the objective alone but on the Lagrangian

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = J(\mathbf{x}) + \sum_{i\in I_1\cup I_2} \lambda_i c_i(\mathbf{x}) \tag{2.2}$$

which also includes the influence of the constraints. In (2.2) $\lambda_i, i \in I_1 \cup I_2$ denote the components of the Lagrangian multiplier $\boldsymbol{\lambda}$. These conditions are known as the *Karush-Kuhn-Tucker conditions* (KKT conditions) and may be stated as follows:

Theorem 2.1 (First order necessary conditions). *Suppose that $\mathbf{x}^*$ is a local solution of* (2.1)*. Additionally assume that the gradients of the active constraints*

$$\{\operatorname{grad} c_i(\mathbf{x}^*) \mid i \in I_1 \vee (i \in I_2 \wedge c_i(\mathbf{x}^*) = 0)\} \tag{2.3}$$

are linearly independent. Then there exists a Lagrangian multiplier $\boldsymbol{\lambda}^*$ with components $\lambda_i^*, i \in I_1 \cup I_2$ such that the following conditions are satisfied:

$$\begin{aligned} \operatorname{grad}_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) &= 0, & & \\ c_i(\mathbf{x}^*) &= 0, & & i \in I_1, \\ c_i(\mathbf{x}^*) &\le 0, & & i \in I_2, \\ \lambda_i^* &\ge 0, & & i \in I_2, \\ \lambda_i^* c_i(\mathbf{x}^*) &= 0, & & i \in I_1 \cup I_2. \end{aligned} \tag{2.4}$$

Proof. Can be found in NOCEDAL AND WRIGHT [115]. ☐

Similar to unconstrained optimization these conditions are only necessary and do not provide any information whether $\mathbf{x}^*$ is a local minimum or not. In unconstrained optimization a necessary condition for $\mathbf{x}^*$ to be a local minimum is that the Hessian of the objective is positive semidefinite. If it is even positive definite, this condition is sufficient for a local minimum. The corresponding conditions for a constrained optimization problem are:

Theorem 2.2 (Second order necessary conditions). *Suppose that the assumptions of Theorem 2.1 are valid, that $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ satisfy the KKT conditions and that $\mathbf{w}$ fulfills*

$$\begin{aligned} \operatorname{grad} c_i(\mathbf{x}^*)^T \mathbf{w} &= 0, & & i \in I_1, \\ \operatorname{grad} c_i(\mathbf{x}^*)^T \mathbf{w} &= 0, & & i \in I_2 \wedge c_i(\mathbf{x}^*) = 0 \wedge \lambda_i^* > 0, \\ \operatorname{grad} c_i(\mathbf{x}^*)^T \mathbf{w} &\le 0, & & i \in I_2 \wedge c_i(\mathbf{x}^*) = 0 \wedge \lambda_i^* = 0. \end{aligned} \tag{2.5}$$

Then

$$\mathbf{w}^T \, \nabla^2_{\mathbf{xx}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) \, \mathbf{w} \geq 0. \tag{2.6}$$

Proof. Can be found in NOCEDAL AND WRIGHT [115]. □

Theorem 2.3 (Second order sufficient conditions). *Suppose, the assumptions of Theorem 2.2 are fulfilled. If for all* $\mathbf{w}$ *satisfying* (2.5)

$$\mathbf{w}^T \, \nabla^2_{\mathbf{xx}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) \, \mathbf{w} > 0, \tag{2.7}$$

then $\mathbf{x}^*$ *is a strict local minimum of* (2.1).

Proof. Can be found in NOCEDAL AND WRIGHT [115]. □

In the following we will give an overview on *Sequential Quadratic Programming* (SQP), a method for effectively solving nonlinear optimization problems. The SQP method generates a sequence of iterates minimizing approximations of (2.1). The key idea is to model (2.1) at the iterate $\mathbf{x}_k$ by a suitable quadratic approximation (the so-called *Quadratic Programming* subproblem) and use the minimizer of this subproblem to define the new iterate $\mathbf{x}_{k+1}$.

In order to motivate the choice of the quadratic subproblem let us forget for a moment the inequality constraints in (2.1). The KKT conditions for this equality constrained optimization problem form a system of nonlinear equations for $(\mathbf{x}, \boldsymbol{\lambda})$ of the form

$$\mathbf{F}(\mathbf{x}, \boldsymbol{\lambda}) = \begin{pmatrix} \operatorname{grad} J(\mathbf{x}) + \mathbf{A}^T(\mathbf{x}) \, \boldsymbol{\lambda} \\ \mathbf{c}(\mathbf{x}) \end{pmatrix} = 0 \tag{2.8}$$

with

$$\mathbf{A}^T(\mathbf{x}) = \begin{pmatrix} \operatorname{grad} c_1(\mathbf{x}) & \cdots & \operatorname{grad} c_l(\mathbf{x}) \end{pmatrix} \tag{2.9}$$

and l denoting the number of equality constraints. Under the assumption that $\mathbf{A}$ has full rank, any minimum of the equality constrained optimization problem satisfies (2.8).

One approach to solve (2.8) is Newton's method. Then, the Newton-step at $(\mathbf{x}_k, \boldsymbol{\lambda}_k)$ fulfills

$$\begin{pmatrix} \mathbf{x}_{k+1} \\ \boldsymbol{\lambda}_{k+1} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_k \\ \boldsymbol{\lambda}_k \end{pmatrix} + \begin{pmatrix} \mathbf{p}_{\mathbf{x}} \\ \mathbf{p}_{\boldsymbol{\lambda}} \end{pmatrix} \tag{2.10}$$

where $\mathbf{p}_{\mathbf{x}}$ and $\mathbf{p}_{\boldsymbol{\lambda}}$ satisfy

$$\begin{pmatrix} \mathbf{W}(\mathbf{x}_k, \boldsymbol{\lambda}_k) & \mathbf{A}^T(\mathbf{x}_k) \\ \mathbf{A}(\mathbf{x}_k) & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}_{\mathbf{x}} \\ \mathbf{p}_{\boldsymbol{\lambda}} \end{pmatrix} = \begin{pmatrix} -\operatorname{grad} J(\mathbf{x}_k) - \mathbf{A}^T(\mathbf{x}_k) \, \boldsymbol{\lambda}_k \\ -\mathbf{c}(\mathbf{x}_k) \end{pmatrix} \tag{2.11}$$

and $\mathbf{W}$ denotes the Hessian of the Lagrangian with respect to $\mathbf{x}$, i.e.

$$\mathbf{W} = \nabla^2_{\mathbf{xx}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}). \tag{2.12}$$

The linear equation system (2.11) together with the update formula (2.10) can be reformulated as

$$\begin{pmatrix} \mathbf{W}(\mathbf{x}_k, \boldsymbol{\lambda}_k) & \mathbf{A}^T(\mathbf{x}_k) \\ \mathbf{A}(\mathbf{x}_k) & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}_{\mathbf{x}} \\ \boldsymbol{\lambda}_{k+1} \end{pmatrix} = \begin{pmatrix} -\operatorname{grad} J(\mathbf{x}_k) \\ -\mathbf{c}(\mathbf{x}_k) \end{pmatrix} \tag{2.13}$$

which can be reinterpreted as KKT-system in the following sense:

$\mathbf{p_x}$ and $\boldsymbol{\lambda}_{k+1}$ solve the first order necessary conditions of the optimization problem

$$\begin{aligned} &\frac{1}{2}\ \mathbf{p}^T\,\mathbf{W}(\mathbf{x}_k,\boldsymbol{\lambda}_k)\,\mathbf{p} + \operatorname{grad} J(\mathbf{x}_k)^T\,\mathbf{p} \to \min_{\mathbf{p}\in\mathbb{R}^n} \\ &\text{subject to} \qquad \mathbf{A}^T(\mathbf{x}_k)\,\mathbf{p} = -\mathbf{c}(\mathbf{x}_k) \end{aligned} \tag{2.14}$$

This quadratic programming (QP) subproblem has a unique solution which can be calculated by solving (2.13) if the second order sufficient conditions are satisfied.

This equivalence between SQP and Newton's method applied to the optimality condition (2.8) is quite useful: For analysis one uses often the Newton point of view, whereas the SQP framework is of advantage for extending the technique also to inequality constraints and for deriving practical algorithms.

The QP framework in (2.14) can be extended straightforward to include also inequality constraints. Then, for problem (2.1) this read as follows:

$$\begin{aligned} &\frac{1}{2}\ \mathbf{p}^T\,\mathbf{W}(\mathbf{x}_k,\boldsymbol{\lambda}_k)\,\mathbf{p} + \operatorname{grad} J(\mathbf{x}_k)^T\,\mathbf{p} \to \min_{\mathbf{p}\in\mathbb{R}^n} \\ \text{subject to} \qquad &\operatorname{grad} c_i(\mathbf{x}_k)\,\mathbf{p} = -c_i(\mathbf{x}_k), \qquad i\in I_1, \\ &\operatorname{grad} c_i(\mathbf{x}_k)\,\mathbf{p} \le -c_i(\mathbf{x}_k), \qquad i\in I_2. \end{aligned} \tag{2.15}$$

This QP subproblem can be solved e.g. by an active set strategy (see e.g. Nocedal and Wright [115] or Gill, Murray, and Wright [63]). If the linearized constraints are inconsistent, i.e. the feasible set of the optimization problem is empty, additional variables are introduced and the objective and the constraints are modified. E.g. in Fletcher [55] the following modification is proposed (the *big-M* method):

$$\begin{aligned} &\frac{1}{2}\ \mathbf{p}^T\,\mathbf{W}(\mathbf{x}_k,\boldsymbol{\lambda}_k)\,\mathbf{p} + \operatorname{grad} J(\mathbf{x}_k)^T\,\mathbf{p} + M(\frac{1}{2}\theta^2+\theta) \to \min_{\mathbf{p}\in\mathbb{R}^n,\theta\in\mathbb{R}} \\ \text{subject to} \qquad &\operatorname{grad} c_i(\mathbf{x}_k)\,\mathbf{p} + (1-\theta)c_i(\mathbf{x}_k) = 0, \qquad i\in I_1, \\ &\operatorname{grad} c_i(\mathbf{x}_k)\,\mathbf{p} + (1-\theta)c_i(\mathbf{x}_k) \le 0, \qquad i\in I_2^+ \\ &\operatorname{grad} c_i(\mathbf{x}_k)\,\mathbf{p} + c_i(\mathbf{x}_k) \le 0, \qquad i\in I_2^-, \end{aligned} \tag{2.16}$$

with $I_2^+ = \{i\in I_2 \mid c_i(\mathbf{x}_k)\ge 0\}$, $I_2^- = I_2\setminus I_2^+$. For this problem $\mathbf{p}=0$, $\theta=1$ denotes a feasible point.

As the SQP method is a variant of Newton's method it is only locally convergent. For an analysis of conditions which guarantee local convergence see e.g. Boggs and Tolle [21]. In order to make it also globally convergent to a local optimum, so-called *globalization strategies* are needed. The two best known ones are *line search methods* and *trust region methods.*

Line search methods introduce a merit function Φ to measure the progress to a solution. Similar to damped Newton's method the calculation of the correction in (2.10) is split into two parts:

- the calculation of a descent direction and
- the calculation of an appropriate step length.

Unlike to unconstrained optimization we can not use the objective itself as a criterion for calculating the step length, but use a merit function which balances the minimization of the objective with the feasibility with respect to the constraints. Popular choices are e.g. the ℓ^1-merit function (cf. HAN [78] or POWELL [122])

$$\Phi(\mathbf{x}) = J(\mathbf{x}) + \frac{1}{\mu}\sum_{i\in I_1}|c_i(\mathbf{x})| + \frac{1}{\mu}\sum_{i\in I_2}[c_i(\mathbf{x})]^+ \tag{2.17}$$

and its variants where $[x]^+ = \max\{x, 0\}$. An alternative is the augmented Lagrangian (cf. SCHITTKOWSKI [136])

$$\Phi(\mathbf{x},\boldsymbol{\lambda}) = J(\mathbf{x}) + \sum_{i\in\bar{I}}(\lambda_i\, c_i(\mathbf{x}) + \frac{1}{2}\,\mu\, c_i(\mathbf{x})^2) - \frac{1}{2\mu}\sum_{i\in I_2\setminus\bar{I}}\lambda_i^2, \tag{2.18}$$

where $\bar{I} = I_1 \cup \{i \in I_2 \mid c_i(\mathbf{x}) \geq \lambda_i\,\mu\}$ and $\boldsymbol{\lambda}$ is an approximation to the Lagrangian multiplier. In both examples μ denotes a scalar penalty parameter. Both merit functions are exact in the sense that for μ sufficiently large minimizers of the original constrained optimization problem also minimize the merit function. For the search direction we solve (2.15) or (2.16), respectively.

The step length has to be chosen in such a way that it ensures a sufficient decrease of the merit function. One set of conditions which guarantee that are the *Armijo-Goldstein-conditions*

$$\Phi(\mathbf{x}_k) + \mu_2\alpha_k\,\langle\Phi_k'(\mathbf{x}_k), \mathbf{p}_k\rangle \leq \Phi(\mathbf{x}_k + \alpha_k\mathbf{p}_k) \leq \Phi(\mathbf{x}_k) + \mu_1\alpha_k\langle\Phi_k'(\mathbf{x}_k), \mathbf{p}_k\rangle \tag{2.19}$$

with $0 < \mu_1 < 1/2 < \mu_2 < 1$ (cf. GILL, MURRAY, AND WRIGHT [63]). When α_k satisfies (2.19) the step is neither too large nor too small. The easiest procedure for calculating a suitable α are simple backtracking procedures starting with $\alpha = 1$ and reducing α in a geometric manner until (2.19) is satisfied. For a more elaborated one using parabolic interpolation see e.g. LUENBERGER [107].

A crucial property in the design of a merit function is that is should accept step length one close to a solution in order to preserve the quadratic convergence of the SQP method. The augmented Lagrangian works well, as long as the estimate of the Lagrangian multiplier is accurate enough, whereas the ℓ^1-merit function sometimes suffers from the so-called *Maratos*-effect, i.e. it does not accept unit step length close to a local minimum and therefore causes a slow-down of the convergence. Strategies to overcome this difficulty using second order corrections can be found e.g. in NOCEDAL AND WRIGHT [115] or CONN, GOULD, AND TOINT [41].

Trust region methods are motivated in a different way. Unlike line search methods which calculate the search direction and the step length in two autonomous steps, trust region methods try to incorporate both aspects into one calculation step. In unconstrained optimization this means that an additional constraint on the maximal increment is added to the quadratic sub-problem which insures that the increment to the iterate is not too large. A straight forward generalization of that approach to constrained optimization problems would lead to

$$\begin{aligned} &\frac{1}{2}\ \mathbf{p}^T\,\mathbf{W}(\mathbf{x}_k,\boldsymbol{\lambda}_k)\,\mathbf{p} + \operatorname{grad} J(\mathbf{x}_k)^T\,\mathbf{p} \to \min_{\mathbf{p}\in\mathbb{R}^m} \\ \text{subject to}\quad &\operatorname{grad} c_i(\mathbf{x}_k)\,\mathbf{p} = -c_i(\mathbf{x}_k), \qquad i \in I_1, \\ &\operatorname{grad} c_i(\mathbf{x}_k)\,\mathbf{p} \leq -c_i(\mathbf{x}_k), \qquad i \in I_2, \\ &\|\mathbf{p}\| \leq \Delta_k. \end{aligned} \tag{2.20}$$

The trust region radius Δ_k is updated in each outer iteration based on comparing the actual decrease in a merit function with the predicted decrease in the QP model. If both are in good agreement the radius is kept or even increased, otherwise it is decreased. The main difficulty of this generalization is that (2.20) need not always have a solution as the admissible set may be empty. That is why other generalizations were developed. One can use e.g. a quadratic model of the augmented Lagrangian merit function instead of the Lagrangian in the QP problem. This results then in

$$\begin{aligned} & \frac{1}{2}\ \mathbf{p}^T\,\mathbf{W}(\mathbf{x}_k,\boldsymbol{\lambda}_k)\,\mathbf{p} + \operatorname{grad} J(\mathbf{x}_k)^T\,\mathbf{p} + \frac{1}{2}\mu\|v\|^2 \to \min_{\mathbf{p}\in\mathbb{R}^n, v\in\mathbb{R}} & & \\ \text{subject to} \quad & \operatorname{grad} c_i(\mathbf{x}_k)\,\mathbf{p} = -c_i(\mathbf{x}_k) + \mu v, & i \in I_1, & \\ & \operatorname{grad} c_i(\mathbf{x}_k)\,\mathbf{p} \le -c_i(\mathbf{x}_k) + \mu v, & i \in I_2, & \qquad (2.21) \\ & \|\mathbf{p}\| \le \Delta_k, & & \end{aligned}$$

which always has a feasible point. Other alternatives can be found e.g. in Conn, Gould, and Toint [41].

2.3 The finite element method

Optimal design problems often incorporate partial differential equations (PDEs) describing the state. As analytic solutions of these PDEs are usually not available, methods for calculating an approximation to the solution of the PDE are needed. Among the most well-known there are

- the *Finite Element Method* (see e.g. Bathe [7], Babuska and Strouboulis [4], Braess [26], Brenner and Scott [29], Ciarlet [39], Hughes [90], Jung and Langer [93], or Zienkiewicz [163]),
- the *Finite Difference Method* (see e.g. Grossmann and Roos [72], or Samarskij [135]),
- the *Finite Volume Method* (see e.g. Grossmann and Roos [72], or Heinrich [83]),
- the *Finite Integration Technique* (see e.g. van Rienen [153], or Weiland [156]),
- or the *Boundary Element Method* (see e.g. Wendland [157], or Chen and Zhou [38]).

Each of these methods has its specific area of application. In the following we want to focus on the finite element method (FEM) and want to explain the basic properties which are needed later on. Furthermore we want restrict ourselves to elliptic boundary value problems for scalar elliptic PDEs of second order of the form

$$\begin{aligned} -\operatorname{div}(A(x)\operatorname{grad} u(x)) + \langle B(x), \operatorname{grad} u(x)\rangle + C(x)u(x) &= f(x), & x &\in \Omega, \\ u(x) &= g(x), & x &\in \Gamma_D, \qquad (2.22) \\ \frac{\partial u}{\partial N}(x) = \langle A(x)\operatorname{grad} u(x), \mathbf{n}(x)\rangle &= h(x), & x &\in \Gamma_N, \end{aligned}$$

where Ω is a bounded domain of $\mathbb{R}^d$ (d is usually 2 or 3) with sufficiently smooth boundary, $\Gamma_D = \overline{\Gamma}_D$, $\Gamma_D \cup \Gamma_N = \partial\Omega$, $\Gamma_D \cap \Gamma_N = \emptyset$. $\mathbf{n}$ denotes the unit outside normal of Ω, $\frac{\partial u}{\partial N}$ the co-normal derivative. Additionally, $A(x)$ is assumed to be symmetric and uniformly positive definite with respect to x. $g(x)$ and $h(x)$ denote the Dirichlet and Neumann boundary data, respectively. Although we restrict ourselves to linear elliptic problems, we want to mention that optimal design problems where the state variable fulfills a transient or nonlinear equation or even a mixed system of equations are important for many practical applications but would make the presentation by far more complicated. We will remark on this where necessary.

In the following we will use the abbreviation

$$\begin{aligned} Lu &= f && \text{in } \Omega, \\ u &= g && \text{on } \Gamma_D, \\ \frac{\partial u}{\partial N} &= h && \text{on } \Gamma_N \end{aligned} \tag{2.23}$$

instead of (2.22). The FEM takes the weak form of the PDE as its starting point. For the problem stated in (2.22) this looks as

$$\begin{aligned} \int\limits_{\Omega} \Big[\langle A(x)\,\mathrm{grad}\,u(x), \mathrm{grad}\,v(x)\rangle + \langle B(x), \mathrm{grad}\,u(x)\rangle v(x) + C(x)u(x)v(x)\Big]\,\mathrm{d}x = \\ \int\limits_{\Omega} f(x)v(x)\,\mathrm{d}x + \int\limits_{\Gamma_N} h(s)v(s)\,\mathrm{d}s \end{aligned} \tag{2.24}$$

or in short

$$a(u,v) = F(v), \qquad \forall\, v \in U \tag{2.25}$$

with the bilinear form $a(u,v)$ and the linear form $F(v)$ defined by the left-hand side and the right-hand side of (2.24), respectively. Up to now it was not specified in which spaces we look for a solution, as well with which functions we test. We could use the Banach space $C^k(\Omega)$, but these turn out to be rather unsuitable for analyzing PDEs. The appropriate spaces are Sobolev-spaces, which are defined in a similar way to C^k but with L^p taking over the role of continuous functions. The appropriate space for stating (2.24), respectively (2.25) is $H^1(\Omega)$ which is defined as

$$H^1(\Omega) = \left\{u \in L^2(\Omega) \mid \mathrm{grad}\,u \in L^2(\Omega)\right\}. \tag{2.26}$$

The complete weak form of (2.21) looks as follows: Find

$$u \in \left\{v \in H^1(\Omega) \mid v = g \text{ on } \Gamma_D\right\}. \tag{2.27}$$

such that

$$a(u,v) = F(v) \qquad \forall\, v \in U = \left\{v \in H^1(\Omega) \mid v = 0 \text{ on } \Gamma_D\right\}. \tag{2.28}$$

The appropriate space for F is U^*, the dual space of U, g must be in the trace space $H^{1/2}(\Gamma_D)$ of $H^1(\Omega)$ on Γ_D. For a complete overview of Sobolev spaces see ADAMS [1].

For showing existence and uniqueness of a solution we transform the variational form (2.25) into homogeneous form by introducing

$$\tilde{u} = u - \tilde{g}$$

with $\tilde{g} \in H^1(\Omega)$ and $\tilde{g} \equiv g$ on Γ_D. Then, $\tilde{u} \in U$ fulfills

$$a(\tilde{u},v) = F(v) - a(\tilde{g},v), \qquad \forall\, v \in U. \tag{2.29}$$

The right-hand side of (2.29) is an element of the dual space of U. The existence and uniqueness of a solution can by shown by a rather simple abstract principle, the so-called *Lax-Milgram lemma.*

Theorem 2.4 (Lax-Milgram lemma). *Let U be a Hilbert space and assume that*

$$a : U \times U \to \mathbb{R} \tag{2.30}$$

is a continuous and elliptic bilinear mapping, i.e. that there exist constants $\alpha, \beta > 0 \in \mathbb{R}$ such that

$$a(u, v) \leq \alpha \, \|u\| \, \|v\|, \qquad \forall \, u, v \in U, \tag{2.31}$$

and

$$a(v, v) \geq \beta \, \|v\|^2, \qquad \forall \, v \in U. \tag{2.32}$$

Finally let $F : U \to \mathbb{R}$ be a bounded linear functional on U, i.e. $F \in U^*$.

Then there exists a unique element $u \in U$ such that

$$a(u, v) = F(v) \qquad \forall \, v \in U. \tag{2.33}$$

Proof. Can be found in Evans [54] or Renardy and Rogers [127]. □

The FEM constructs an approximation to the solution u of the variational equality (2.28). Therefore the solution space as well as the test space are approximated by a sequence of finite dimensional subspaces which are constructed as follows:

Let τ_h be a regular triangulation (c.f. Ciarlet [39]) of the domain Ω into geometric elements, e.g. triangles or quadrilaterals in 2D, tetrahedra or hexahedra in 3D. On these geometric elements we define so called finite elements using shape functions with local support, e.g. linear, bilinear or quadratic shape functions. Therewith we define solution and test spaces e.g. with piecewise linear, bilinear or quadratic functions. The discretization parameter h of the family of finite dimensional subspaces is usually related to the mesh-size of the triangulation. Additionally we impose that the FE-spaces are nested, i.e.

$$V_{h_1} \subseteq V_{h_2} \quad \text{if } h_1 \leq h_2 \tag{2.34}$$

and that this family of FE-space is complete in the limit, i.e.

$$\overline{\bigcup_{h>0} V_h} = H^1(\Omega). \tag{2.35}$$

In order to calculate an approximation to the solution of (2.28) we need finite dimensional approximations of the solution and the test space in (2.28). The finite dimensional approximation of the solution space in (2.28) is

$$U_h^g = \{ v_h \in V_h \mid v_h = g_h \text{ on } \Gamma_D \}$$

where $g_h \in V_h$ is an approximation of g on Γ_D. The finite dimensional approximation of the test space is

$$U_h = \{ v_h \in V_h \mid v_h = 0 \text{ on } \Gamma_D \} .$$

By replacing the solution and the test space in (2.28) by their finite dimensional analogs we get a finite dimensional approximation of (2.28) which results in

$$a(u_h, v_h) = F(v_h), \qquad \forall \, v_h \in U_h, \tag{2.36}$$

with $u \in U_h^g$. For the solution of (2.36) we again homogenize the variational equality. Choosing a basis

$$\Phi = (\phi_1, \ldots, \phi_m)^T \in V_h \tag{2.37}$$

of the finite dimensional subspace U_h, we may represent $u_h \in U_h^g$ via

$$u_h = \mathbf{u}^T \Phi + g_h \tag{2.38}$$

with a coefficient vector $\mathbf{u} \in \mathbb{R}^m$. In order to transform (2.36) into a linear system of equations, we define the stiffness matrix

$$\mathbf{K} = \big(a(\phi_j, \phi_i)\big)_{i,j=1,\ldots,m} \tag{2.39}$$

and the load vector

$$\mathbf{f} = \big(F(\phi_j) - a(g_h, \phi_j)\big)_{j=1,\ldots,m} \quad . \tag{2.40}$$

This allows us to write the discretized variational problem (2.36) as

$$\mathbf{K}\,\mathbf{u} = \mathbf{f}. \tag{2.41}$$

In the following section we want to analyze the properties of the stiffness matrix $\mathbf{K}$ and show efficient algorithms for solving (2.41).

2.4 Iterative solvers and preconditioning

Before we begin the discussion on efficient solution methods for linear systems of equations as in (2.41) we repeat the basic properties of the system matrix $\mathbf{K}$. Furthermore we introduce the basic notation used below.

In many practical applications the bilinear form $a(\cdot,\cdot)$ in (2.36) is symmetric, that is why we restrict our presentations to this case. Additionally we want to assume that the propositions of the Lax-Milgram lemma are fulfilled, i.e. the continuous system is well-posed.

Due to the construction of the FE spaces in Section 2.3 the number of base functions in (2.37) is large, typically the dimension of $\mathbf{K}$ is asymptotically

$$\dim(\mathbf{K}) = \mathcal{O}(h^{-d}) \tag{2.42}$$

where the discretization parameter h denotes the typical average mesh size and d the space dimension of the computational domain. Fortunately, the matrix is sparse. As the finite element base functions have only local support most entries in $\mathbf{K}$ vanish, i.e. the system matrix is sparse. For a regular triangulation only a fixed number of entries are nonzero in each line, i.e. the total number of nonzero entries is proportional to the dimension of the matrix. That is why, a matrix-vector multiplication can be realized with $\mathcal{O}(h^{-d})$ multiplications.

Due to the assumptions on the bilinear form made above, the stiffness matrix $\mathbf{K}$ is symmetric. Furthermore, the discrete system (2.41) is well-posed, i.e. $\mathbf{K}$ is regular. But, $\mathbf{K}$ is even positive definite, as $a(\cdot,\cdot)$ fulfills the ellipticity condition (2.32), i.e. the eigenvalues $\mu_i(\mathbf{K}), i = 1, \ldots, m$ of $\mathbf{K}$ are real and

$$0 < \mu_1(\mathbf{K}) \leq \cdots \leq \mu_m(\mathbf{K}). \tag{2.43}$$

This can be exploited by solution algorithms as we will see later. The corresponding eigenvectors $\varphi_i(\mathbf{K}), i = 1, \ldots, m$ form an ortho-normal system, i.e.

$$\langle \varphi_i(\mathbf{K}), \varphi_j(\mathbf{K}) \rangle_2 = \delta_{ij} \tag{2.44}$$

where $\langle\cdot,\cdot\rangle_2$ denotes the Euclidian scalar product and δ_{ij} the Kronecker symbol.

The spectral condition number of the stiffness matrix, defined by

$$\kappa(\mathbf{K}) = \frac{\mu_m(\mathbf{K})}{\mu_1(\mathbf{K})}, \tag{2.45}$$

is very large as h tends to zero, for the discretization of a PDE of second order

$$\kappa(\mathbf{K}) = \mathcal{O}(h^{-2}). \tag{2.46}$$

We want to solve a sparse linear system of equations with a symmetric, positive definite matrix of very large dimension with a rather large condition number.

There are two different approaches for solving linear systems of equation: Either using direct methods or using iterative ones.

Direct methods for solving sparse linear systems perform a Cholesky or Gauss factorization of the matrix, i.e. they look for a representation of the form

$$\mathbf{K} = \mathbf{U}^T\,\mathbf{U}$$

or

$$\mathbf{K} = \mathbf{L}\,\mathbf{U},$$

respectively. $\mathbf{L}$ is a lower triangular matrix where all diagonal elements are 1, and $\mathbf{U}$ denotes an upper triangular matrix. For this factorization, sparse direct solvers try to reduce the costs by reordering rows and columns in such a way that the fill-in is minimized, i.e. they try to minimize the number of nonzero elements introduced by the factorization process at positions where the original matrix was zero. Typically, a sparse direct solver consists of the following phases:

- First, a column pre-ordering solely on the nonzero pattern of the matrix is performed. This is done in such a way that the factorization of the reordered matrix is as sparse as possible. Unfortunately, this problem is NP-hard (Yannakakis [159]), so heuristics are used. Two popular methods are minimum degree ordering and nested-dissection ordering.

- Then the matrix is factorized. In the symmetric and positive definite case, first a symbolic factorization using only the matrix pattern is done to get the matrix pattern of the factorization and then the factors are calculated. In this case, the column renumbering of the first phase is also used for renumbering the rows, since any symmetric permutation of the rows and columns will be numerically acceptable. If the matrix is not symmetric and positive definite, the calculation of the pattern of the factors can not be separated from the factorization itself. In this case pivoting is necessary which is done without regard to sparsity.

- Finally forward and backward triangular sweeps are executed.

Two references to this topic are the books by George and Liu [57] or Duff, Erisman, and Reid [47]. These techniques have also been implemented e.g. in Sparskit [131] or SuperLU [45].

Algorithm 2.1 Preconditioned Richardson iteration

Define a damping parameter τ, $0 < \tau < \frac{2}{c_2}$

Initialize start value $\mathbf{u}_0$
$i = 0$

while not converged **do**
  $\mathbf{u}_{i+1} = \mathbf{u}_i + \tau \mathbf{C}^{-1} (\mathbf{f} - \mathbf{K}\, \mathbf{u}_i)$
  $i = i + 1$
end while

Iterative methods for solving large sparse linear systems have been gaining popularity during the last decade. Until recently direct methods were often preferred by engineers, especially in real life applications due to their robustness and predictable behavior. However, direct solvers are inefficient, not only the computational effort but also the memory consumption is very high, especially for very large sparse matrices. On the other hand, iterative solvers exploit sparsity to a much higher extent. They mainly need matrix-vector multiplications and other operations whose calculation time is proportional to the number of unknowns. Iterative solvers are usually efficient only when combined with appropriate preconditioners. That is why we want to treat both aspects together. For the rest of this section we want to assume that the symmetric and positive definite preconditioner $\mathbf{C}$ is spectrally equivalent to the matrix $\mathbf{K}$, i.e.

$$\exists c_1, c_2 \in \mathbb{R}^+ \qquad c_1 \langle \mathbf{Cu}, \mathbf{u} \rangle \leq \langle \mathbf{Ku}, \mathbf{u} \rangle \leq c_2 \langle \mathbf{Cu}, \mathbf{u} \rangle, \qquad \forall \mathbf{u} \in \mathbb{R}^m, \tag{2.47}$$

with positive spectral equivalence constants c_1, c_2. A preconditioner is constructed in such a way that the quotient $\frac{c_2}{c_1}$ is as small as possible under the restriction that the preconditioning operation $\mathbf{C}^{-1}\, \mathbf{d}$ is efficiently evaluable. The optimal choices for the spectral equivalence constants are

$$\begin{aligned} c_1 &= \mu_{\min}(\mathbf{C}^{-1}\, \mathbf{K}) \\ c_2 &= \mu_{\max}(\mathbf{C}^{-1}\, \mathbf{K}), \end{aligned} \tag{2.48}$$

where $\mu_{\min}(\mathbf{C}^{-1}\, \mathbf{K})$ and $\mu_{\max}(\mathbf{C}^{-1}\, \mathbf{K})$ denote the minimal and maximal eigenvalue of $\mathbf{C}^{-1}\, \mathbf{K}$ respectively, which are both real. In the following we will abbreviate the spectral equivalence (2.47) by

$$c_1\, \mathbf{C} \leq \mathbf{K} \leq c_2\, \mathbf{C}. \tag{2.49}$$

As $\mathbf{K}$ is assumed to be symmetric and positive definite, (2.41) is a necessary and sufficient criterion for a minimizer of the energy functional

$$\frac{1}{2}\, \mathbf{u}^T \mathbf{K}\, \mathbf{u} - \mathbf{f}^T \mathbf{u} \to \min_{\mathbf{u} \in \mathbb{R}^m} . \tag{2.50}$$

That is why, many iteration schemes can be motivated as minimization algorithms for (2.50).

The easiest method is a variant of the steepest descent method where the step length is chosen a-priorily. This iteration is usually named Richardson iteration. A preconditioned version can be found in Algorithm 2.1. The error iteration scheme can be formulated as

$$\mathbf{u}^{i+1} - \mathbf{u} = (\mathbf{I} - \tau \mathbf{C}^{-1} \mathbf{K})(\mathbf{u}^i - \mathbf{u}) \tag{2.51}$$

Algorithm 2.2 Preconditioned steepest descent iteration

Initialize start value $\mathbf{u}_0$
$i = 0$

while not converged **do**
  /* *Calculate descent direction* */
  $\mathbf{r}_i = \mathbf{f} - \mathbf{K}\,\mathbf{u}_i$
  $\mathbf{s}_i = \mathbf{C}^{-1}\,\mathbf{r}_i$
  /* *Calculate step length* */
  $\tau_i = \frac{\langle \mathbf{r}_i, \mathbf{s}_i \rangle}{\langle \mathbf{s}_i, \mathbf{K}\,\mathbf{s}_i \rangle}$
  /* *Update iterate* */
  $\mathbf{u}_{i+1} = \mathbf{u}_i + \tau_i \mathbf{s}_i$
  $i = i + 1$
end while

where $\mathbf{I} \in \mathbb{R}^{m \times m}$ denotes the identity matrix. It can be seen, that the step length does not depend on the iteration. This implies that the iteration converges if and only if

$$\rho(\mathbf{I} - \tau \mathbf{C}^{-1}\mathbf{K}) < 1. \tag{2.52}$$

The spectral radius ρ of the iteration matrix can be bounded using the spectral equivalence constants c_1, c_2 by

$$\rho(\mathbf{I} - \tau \mathbf{C}^{-1}\mathbf{K}) \leq \max\{|1 - \tau c_1|, |1 - \tau c_2|\} \tag{2.53}$$

i.e. the iteration converges for $0 < \tau < 2/c_2$. The right hand side of the above inequality is minimal for $\tau = \frac{1}{c_1 + c_2}$ which leads to a convergence rate of

$$\rho(\mathbf{I} - \tau \mathbf{C}^{-1}\mathbf{K}) \leq \frac{\kappa(\mathbf{C}^{-1}\mathbf{K}) - 1}{\kappa(\mathbf{C}^{-1}\mathbf{K}) + 1} \leq \frac{c_2 - c_1}{c_2 + c_1}. \tag{2.54}$$

An improved version is the steepest descent method. There, the step length parameter is no more chosen a-priorily, but depends on the iteration. A preconditioned version can be found in Algorithm 2.2.

A method which is more elaborated is the conjugate gradient (CG) method, developed by Hestenes and Stiefel [85]. It is one of the best known iterative techniques for solving sparse symmetric positive definite linear systems. It is a representative of a larger class of methods, the Krylov-subspace-correction methods. The basic idea of these methods is to minimize the defect in the Krylov-subspace

$$\mathcal{K}_k(\mathbf{A}, \mathbf{y}) = \left\{ \mathbf{y}, \mathbf{A}\mathbf{y}, \ldots \mathbf{A}^{k-1}\mathbf{y} \right\} \tag{2.55}$$

generated by a matrix $\mathbf{A}$ and a vector $\mathbf{y}$ in the k-th iteration step. The preconditioned CG-method takes as generating matrix $\mathbf{C}^{-1/2}\mathbf{K}\,\mathbf{C}^{-1/2}$, and as generating vector the initial residual. The algorithm is presented in Algorithm 2.3. Compared to the Richardson iteration it can be seen that the CG algorithm is no more a linear method but it is nonlinear. Furthermore it is not necessary to provide a suitable damping factor for which knowledge or some bounds on the extreme eigenvalues of $\mathbf{C}^{-1/2}\mathbf{K}\,\mathbf{C}^{-1/2}$ is needed. The CG iteration

Algorithm 2.3 Preconditioned conjugate gradient iteration

Initialize start value $\mathbf{u}_0$
$\mathbf{r}_0 = \mathbf{f} - \mathbf{K}\,\mathbf{u}_0$
$\mathbf{d}_0 = \mathbf{C}^{-1}\,\mathbf{r}_0$
$\mathbf{s}_0 = \mathbf{d}_0$
$i = 0$

while not converged **do**
  $\gamma_i = \langle \mathbf{s}_i, \mathbf{r}_i \rangle$
  $\alpha_i = \frac{\gamma_i}{\langle \mathbf{K}\,\mathbf{d}_i, \mathbf{d}_i \rangle}$
  $\mathbf{u}_{i+1} = \mathbf{u}_i + \alpha_i \mathbf{d}_i$
  $\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i\,\mathbf{K}\,\mathbf{d}_i$
  $\mathbf{s}_{i+1} = \mathbf{C}^{-1}\,\mathbf{r}_{i+1}$
  $\beta_i = \frac{\gamma_i}{\langle \mathbf{s}_{i+1}, \mathbf{d}_i \rangle}$
  $\mathbf{d}_{i+1} = \mathbf{s}_{i+1} + \beta\,\mathbf{d}_i$
  $i = i + 1$
end while

calculates an optimal damping factor by itself using the underlying minimization property. The convergence can be estimated in the $\mathbf{K}$-energy norm

$$\|\mathbf{y}\|_{\mathbf{K}} = \langle \mathbf{y}, \mathbf{K}\,\mathbf{y} \rangle^{1/2}. \tag{2.56}$$

The error in the k-th iteration step of the preconditioned CG algorithm is bounded by

$$\|\mathbf{u}^k - \mathbf{u}\|_{\mathbf{K}} \le q_k\, \|\mathbf{u}^0 - \mathbf{u}\|_{\mathbf{K}} \tag{2.57}$$

with

$$q_k = \frac{c^k}{1 + c^{2k}} \qquad \text{with} \qquad c = \frac{\sqrt{\kappa(\mathbf{C}^{-1}\mathbf{K})} - 1}{\sqrt{\kappa(\mathbf{C}^{-1}\mathbf{K})} + 1} \le \frac{\sqrt{\frac{c_2}{c_1}} - 1}{\sqrt{\frac{c_2}{c_1}} + 1} \tag{2.58}$$

(see e.g. Hackbusch [77], Meurant [112], Axelsson [3], or Jung and Langer [93] for details). c_1, c_2 denote the spectral equivalence constants (see (2.49)). It can be seen that this bound is similar to that of the Richardson iteration, except that the condition number of $\mathbf{C}^{-1}\mathbf{K}$ is replaced by its square root.

For both algorithms we get the unpreconditioned versions by taking $\mathbf{C}$ to be the identity matrix. But it can be seen immediately that the unpreconditioned iterations converge very slowly because the convergence rate is very close to 1 for large condition numbers $\kappa(\mathbf{K})$. That is why the choice of effective preconditioners is very important. In the following we want to motivate properties for good preconditioners.

From (2.58) it is immediately clear that one has to take $\mathbf{C} = \mathbf{K}$ to minimize the convergence rate, which is of no help due to the effort for inverting $\mathbf{K}$. In order to get good convergence properties the quotient c_2/c_1 shall be close to one. In practice, it should be at least independent or almost independent of the mesh-size h. Nevertheless applying $\mathbf{C}^{-1}$ should not be too much effort. The following properties turned out to be very useful:

- The condition number $\kappa(\mathbf{C}^{-1}\mathbf{K})$ shall be close to 1. Additionally it should be bounded from above independently of the mesh-size parameter h of the discretization.

- The computational effort of the preconditioning operation $\mathbf{C}^{-1}$ should be not too high, if possible proportional to a multiplication with $\mathbf{K}$.
- The memory requirements for realizing the preconditioning step should be comparable with the ones needed for a multiplication with $\mathbf{K}$.

No preconditioning, i.e. $\mathbf{C} = \mathbf{I}$ clearly fulfills the latter two requirements, but the condition number $\kappa(\mathbf{C}^{-1}\mathbf{K})$ behaves like $\mathcal{O}(h^{-2})$ which implies that the iteration converges extremely slowly. Classical iteration procedures like the Jacobi- or the Gauß-Seidel method do not improve this behavior. On the other hand, taking $\mathbf{C} = \mathbf{K}$ results in a direct solver, for which the first property is clearly fulfilled. Nevertheless it is in general impossible to fulfill the other two.

Multigrid preconditioners have shown to fulfill all 3 properties (see e.g. Hackbusch [76] or Jung and Langer [92]). They are based on a well-balanced interplay between a smoothing operator and a coarse grid correction and can be motivated in the following way:

Let's consider the error after k iterations

$$\mathbf{z}^k = \mathbf{u}^k - \mathbf{u} \tag{2.59}$$

and develop it into a Fourier sequence taking the eigenfunctions of $\mathbf{K}$ as basis. From the analysis of classical iteration methods like Jacobi of Gauß-Seidel it it known, that they reduce the high-frequency components of the error very fast (Hackbusch [77]) whereas they have problems to reduce the low frequency components of the error. That is why, the smooth part of the error (after smoothing the high-frequency components on the fine grid) is approximated on a coarser grid. The corresponding equation system on the coarser grid can be solved easier, as it contains by far fewer unknowns. Furthermore the idea can be applied recursively which leads to the multigrid iteration presented in Algorithm 2.4. The key-point is the efficient interplay between the smoothing operator and the coarse grid correction.

The idea presented above is based on nested finite element spaces which are needed for the definition of the coarse grid system. These are often not available due to limitations of the computer program or the coarsest equation system is still too large for an efficient application of a sparse direct solver. Then, algebraic multigrid methods can be applied. These mimic the coarse grid and define smoothing and prolongation operators using only the stiffness matrix itself or using only information on the finest grid. An overview of standard multigrid methods as well as algebraic multigrid methods is presented in Trottenberg, Oosterlee, and Schüller [152]. For publications focusing more on algebraic multigrid methods see Reitzinger [126] or Ruge and Stüben [130] and references therein.

Algorithm 2.4 Basic concept of a symmetric multigrid V-cycle MgStep (ℓ, $\mathbf{K}_\ell$, $\mathbf{f}_\ell$, $\mathbf{u}_\ell$)

/ ℓ denotes the level, $\ell = 1$ is the coarsest level */*

if $\ell == 1$ **then**
  solve the system on the coarsest grid, $\mathbf{u}_\ell = \mathbf{K}_\ell^{-1}\,\mathbf{f}_\ell$
else
  / Pre-smoothing: */*
  / $\mathcal{S}$ denotes the smoothing operator */*
  / ν denotes the number of smoothing steps */*
  $\mathbf{u}_\ell = \mathcal{S}_\ell^\nu(\mathbf{u}_\ell, \mathbf{f}_\ell)$
  / Defect calculation */*
  $\mathbf{d}_\ell = \mathbf{f}_\ell - \mathbf{K}_\ell\,\mathbf{u}_\ell$
  / Restriction onto coarser grid */*
  / $\mathcal{P}_\ell$ denotes the prolongation operator between level $\ell - 1$ and ℓ */*
  $\mathbf{d}_{\ell-1} = \mathcal{P}_\ell^T\,\mathbf{d}_\ell$
  / Solve coarse grid system */*
  / $\mathbf{K}_{\ell-1}$ denotes the stiffness matrix on the coarser grid */*
  MgSTEP($\ell - 1$, $\mathbf{K}_{\ell-1}$, $\mathbf{d}_{\ell-1}$, $\mathbf{w}_{\ell-1}$)
  / Prolongation from the coarser mesh to the current one */*
  $\mathbf{w}_\ell = \mathcal{P}_\ell\,\mathbf{w}_{\ell-1}$
  / Add coarse grid correction */*
  $\mathbf{u}_\ell = \mathbf{u}_\ell + \mathbf{w}_\ell$
  / Post-smoothing */*
  $\mathbf{u}_\ell = (\mathcal{S}_\ell^\nu)^T(\mathbf{u}_\ell, \mathbf{f}_\ell)$
end if

Chapter 3

Automatic Differentiation, an Introduction

3.1 Motivation

During the last decades the simulation tools used in product design became faster and more and more elaborated. Therefore, the interest on sensitivities of the output with respect to changes in the design or other input quantities (e.g. material parameters) increased. The main problem, which arose, resulted from the fact that meanwhile good simulation tools were available, but non of the program designers ever thought on derivatives during the development of the simulation tool. Still nowadays people often use finite differences (see e.g. HAASE AND LINDNER [73]) due to this fact.

One alternative to finite differences would be to use computer algebra and symbolic methods for generating code to calculate derivatives. Symbolic methods take a closed form representation of a formula as their starting point. Usually this is of the form

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) \qquad \text{with } \mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m. \tag{3.1}$$

These methods do not use any information on how to evaluate $\mathbf{f}$ and which intermediate results to store. They apply the elementary rules of calculus to $\mathbf{f}$, get a huge expression representing the derivative and try to simplify this expression using algebraic manipulation. Typical examples for software packages of that kind are MAPLE [125] or MATHEMATICA [158]. This approach works fine for small examples but reaches its bounds for more complicated ones or when higher order derivatives are needed. Table 3.1 contains the complexity of a C-code to evaluate the calculated derivatives (the numbers were taken from presentation notes of A. Griewank). As can be seen, the complexity of the code for calculating the derivative explodes exponentially which makes a symbolic approach completely unattractive if not even impossible.

During the same time a completely different and not so well known scientific community grew up, which mainly focused on doing derivatives. This area is now called *Automatic differentiation* or *Algorithmic differentiation*, short AD. Their approach was completely different to the one in computer algebra and symbolic computation. In AD the starting point is a computer program to evaluate the function. This computer program can also be seen as a closed form representation, nevertheless it is completely different to the input of computer

derivative order	lines of C-Code
0	1
1	2
2	49
3	421
4	4164
5	57027

Table 3.1: C code generated by Mathematica (`CForm`), courtesy by A. Griewank

algebra packages. In a computer program the implementing person grouped several statements together to calculate intermediate results. These intermediate results are then used to calculate new intermediate results or the function value, respectively. When implementing a derivative there is no need to propagate all the symbolic derivative information of the elementary functions into one final formula. It is by far better to exploit the structure using the intermediate results generated in a computer program also to calculate derivatives.

In AD two different ways to calculate derivatives were developed: On the one hand the *forward mode* which is comparable to directional derivatives, on the other hand the *reverse mode* which can be compared to gradient evaluation. Both are explained in more detail in Section 3.3 and Section 3.4, respectively. At the beginning there is a short introduction into the framework and notation used in the following sections. We use the same notation as in the monograph by GRIEWANK [67] which can be seen as the standard reference on this topic.

3.2 Framework and notation

On a very basic level every routine for evaluating a function can be subdivided into 3 components:

- Copy the values of the independent variables to internal variables.
- Evaluate the body of the function using only internal variables for storing intermediate results.
- Assign the internal variables containing the results to the dependent variables.

As the first and the last part contain only assignments it is enough to analyze the middle-section with uses only internal variables. This variable vector can be written as

$$[\underbrace{v_{1-n},\ldots,v_0}_{\mathbf{x}},v_1,\ldots,v_{l-m},\underbrace{v_{l-m+1},\ldots,v_l}_{\mathbf{y}}]. \tag{3.2}$$

Each value v_i is obtained by applying an elementary function ϕ_i to some set of arguments v_j with $j < i$, so that we can write

$$v_i = \phi_i\big((v_j)_{j\prec i}\big). \tag{3.3}$$

Some elementary functions are listed in Table 3.2. For a more complete list see GRIEWANK [67, p. 25]. For simplicity we restrict our attention to smooth functions here. For real life situations also Lipschitz continuous function like e.g. $|u|$, $\min(u,v)$, $\|(u_1,\ldots,u_n)\|_2$, $\max(u,v)$ or even

Essential	Optional
$u+v$, $u \cdot v$	$u-v$
$-u$, c	$c \cdot u$, $c \pm u$
$1/u$	u/v
$\exp(u)$, $\log(u)$	u^c
$\sin(u)$, $\cos(u)$	$\tan(u)$, $\arcsin(u)$
u, v variables, c constant	

Table 3.2: Examples for elementary functions

non-smooth functions like e.g. the sign of a number $\mathrm{sign}(u)$ or the Heaviside-function $\mathrm{heav}(u)$ are needed (e.g. to represent branches in the control flow). For a treatment of these functions see Griewank [67, Chapter 11].

The dependence relation $j \prec i$ in (3.3) means that v_i depends directly only on some of the $v_j, j < i$, typically on one or two indices $j < i$. This dependence relation defines a partial ordering of all indices $i = 1-n, \ldots, l$. The relations of the variables $v_i, i = 1-n, \ldots, l$ can be visualized in a computational graph. This is an acyclic graph in which the vertices define variables and an arc runs from v_j to v_i if and only if $j \prec i$. The roots of the graph represent the independent variables, the leafs the dependent ones.

To summarize the considerations the general evaluation procedure of a function can be seen in Algorithm 3.1. We highlighted the three evaluation phases using comment statements in a C-like syntax. In the following sections we will use this general evaluation procedure to show how directional derivatives and gradients can be implemented in principle.

In order to make this notation clearer let us consider the following example taken from Griewank [67]. This example will also be used for the illustration of the forward and reverse mode in the following sections.

Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined as

$$y = \left(\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - \exp(x_2)\right) \cdot \left(\frac{x_1}{x_2} - \exp(x_2)\right). \tag{3.4}$$

The evaluation procedure for this simple formula can be seen in Algorithm 3.2. The three dis-

Algorithm 3.1 General evaluation procedure of a function

/* *Assignment phase* */
for $i = 1$ to n **do**
  $v_{i-n} = x_i$
end for
/* *Calculation phase* */
for $i = 1$ to l **do**
  $v_i = \phi_i\big((v_j)_{j \prec i}\big)$
end for
/* *Assignment phase* */
for $i = m-1$ to 0 step -1 **do**
  $y_{m-i} = v_{l-i}$
end for

Algorithm 3.2 Evaluation trace of a simple model example

/* *Assignment phase* */
$v_{-1} = x_1 = 1.5000$
$v_0 = x_2 = 0.5000$
/* *Calculation phase* */
$v_1 = v_{-1}/v_0 = 1.5000/0.5000 = 3.0000$
$v_2 = \sin(v_1) = \sin(3.0000) = 0.1411$
$v_3 = \exp(v_0) = \exp(0.5000) = 1.6487$
$v_4 = v_1 - v_3 = 3.000 - 1.6487 = 1.3513$
$v_5 = v_2 + v_4 = 0.1411 + 1.3513 = 1.4924$
$v_6 = v_5 \cdot v_4 = 1.4924 \cdot 1.3513 = 2.0167$
/* *Assignment phase* */
$y = v_6 = 2.0167$

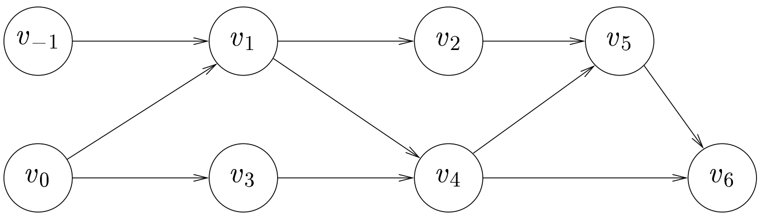

Figure 3.1: Computational graph of a simple model example

tinct phases of the evaluation can be seen clearly, the first phase containing only assignments, the calculation phase and finally the phase assigning the output variables. A computational graph of the calculation phase can be found in Figure 3.1. We see that the evaluation of the formula is not done straightforward, but in a clever way by eliminating common sub-expressions and using internal variables for them. As we will see, this is one of the secrets to reduce the evaluation effort for derivatives compared to the evaluation of a derivative generated by a symbolic computation package. We want to reuse this very simple example to illustrate how the forward and the reverse mode calculate derivatives.

3.3 The forward mode – Propagation of tangents

In this section we want to calculate directional derivatives, also called tangents of functions. It is assumed that the function can be decomposed into a sequence of elementary functions which are continuously differentiable. In contrast to approximating the derivative by finite differences, we obtain truncation error free numerical values for the derivative by using the method presented here.

For the forward mode let us consider a function

$$\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m, \quad \mathbf{x} \mapsto \mathbf{f}(\mathbf{x}). \tag{3.5}$$

As we want to calculate a directional derivative, let us further introduce a curve

$$\mathbf{x} : \mathbb{R} \to \mathbb{R}^n, \quad t \mapsto \mathbf{x}(t) \tag{3.6}$$

and define the resulting curve

$$\mathbf{y} : \mathbb{R} \to \mathbb{R}^m, \quad t \mapsto \mathbf{f}(\mathbf{x}(t)). \tag{3.7}$$

According to the chain rule the derivative of $\mathbf{y}$ is defined by

$$\dot{\mathbf{y}}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{f}(\mathbf{x}(t)) = \mathbf{f}'(\mathbf{x}(t)) \cdot \dot{\mathbf{x}}(t), \tag{3.8}$$

where $\mathbf{f}' \in \mathbb{R}^{m\times n}$ denotes the Jacobian of $\mathbf{f}$. $\dot{\mathbf{x}}$ can be interpreted as direction at the point $\mathbf{x}(t)$ and $\dot{\mathbf{y}}$ denotes the directional derivative of $\mathbf{f}$ into the direction of $\dot{\mathbf{x}}$ then. From a geometrical point of view $\dot{\mathbf{x}}$ and $\dot{\mathbf{y}}$ can be interpreted as tangents of $\mathbf{x}(t)$ respectively $\mathbf{y}(t)$. $\mathbf{f}'$ maps tangents along curves in the definition space of $\mathbf{f}$ onto tangents of curves in its image space. That is why this approach is also called propagation of tangents.

Let us now assume that $\mathbf{f}$ can be decomposed into elementary functions as described in Section 3.2, i.e.

$$\mathbf{f}(x) = (\phi_l \circ \phi_{l-1} \circ \cdots \circ \phi_2 \circ \phi_1)(x). \tag{3.9}$$

Then using the chain rule one obtains

$$\mathbf{f}'(x) \cdot \dot{\mathbf{x}} = \phi_l' \cdot \phi_{l-1}' \cdot \cdots \cdot \phi_2' \cdot \phi_1' \cdot \dot{\mathbf{x}} \tag{3.10}$$

for its derivative, which can be evaluated as

$$\mathbf{f}'(\mathbf{x}) \cdot \dot{\mathbf{x}} = \phi_l' \cdot \left(\phi_{l-1}' \cdot \left(\cdots \left(\phi_2' \cdot \left(\phi_1' \cdot \dot{\mathbf{x}}\right)\right) \cdots\right)\right) \tag{3.11}$$

due to associativity. In the framework of Section 3.2 the evaluation procedure for (3.11) can be described as in Algorithm 3.3. Assembling the complete Jacobian is not needed and usually

Algorithm 3.3 General evaluation procedure of the directional derivative of a function

/* *Assignment phase* */
for $i = 1$ to n **do**
  $v_{i-n} = x_i$
  $\dot{v}_{i-n} = \dot{x}_i$
end for
/* *Calculation phase* */
for $i = 1$ to l **do**
  $v_i = \phi_i\big((v_j)_{j\prec i}\big)$
  $\dot{v}_i = \sum_{j\prec i} \frac{\partial}{\partial v_j}\phi_i\big((v_k)_{k\prec i}\big) \cdot \dot{v}_j$
end for
/* *Assignment phase* */
for $i = m-1$ to 0 step -1 **do**
  $y_{m-i} = v_{l-i}$
  $\dot{y}_{m-i} = \dot{v}_{l-i}$
end for

quite uneconomical except when very many different directional derivatives at a fixed point $\mathbf{x}$ are needed. In this directional derivative procedure $\dot{v}_i$ can be interpreted as the directional derivative of v_i into the direction $\dot{\mathbf{x}}$. The main difference of the AD approach compared to

ϕ	$[\phi, \dot{\phi}]$
$w = c$	$w = c$ $\dot{w} = 0$
$w = u \pm v$	$w = u \pm v$ $\dot{w} = \dot{u} \pm \dot{v}$
$w = u \cdot v$	$\dot{w} = \dot{u} \cdot v + u \cdot \dot{v}$ $w = u \cdot v$
$w = 1/u$	$w = 1/u$ $\dot{w} = -w \cdot w \cdot \dot{u}$
$w = \sqrt{u}$	$w = \sqrt{u}$ $\dot{w} = 0.5 \cdot \dot{u}/w$
$w = u^c$	$\dot{w} = \dot{u}/u$ $w = u^c$ $\dot{w} = c \cdot w \cdot \dot{w}$
$w = \exp(u)$	$w = \exp(u)$ $\dot{w} = w \cdot \dot{u}$
$w = \log(u)$	$\dot{w} = \dot{u}/u$ $w = \log(u)$
$w = \sin(u)$	$\dot{w} = \cos(u) \cdot \dot{u}$ $w = \sin(u)$

Table 3.3: Elementary functions and their tangent statements

the symbolic calculation of derivatives is the fact that numerical values of the derivatives are propagated rather than their corresponding symbolic expressions.

In Algorithm 3.3 we placed the tangent statement to calculate $\dot{v}_i$ after the original statement as this order of calculation seems quite natural. As long as two variables do not overwrite it is not necessary to think of a special execution order of the statements calculating the function value and its derivative. However, in real programs several variables v_j often share one common storage location. Especially when v_i shares the storage location with one of the arguments v_j of ϕ_i it is necessary to update $\dot{v}_i$ before v_i is updated. That is why, source to source transformers (see also Section 3.5) always put the derivative statement ahead of the original one. On the other hand the value of the derivative $\dot{v}_i$ can often be calculated easier when v_i is already known. Therefore ϕ_i and $\dot{\phi}_i$ should be evaluated simultaneously sharing intermediate results. The most elementary cases are listed in Table 3.3. It is easy to observe that for the computation of $\dot{\mathbf{y}}$ each elementary function is processed exactly once. With respect to the computational complexity of evaluating $\dot{\mathbf{y}}$ this implies the following:

Theorem 3.1. *Assume that for each ϕ_i the effort for evaluating $\phi_i' \cdot \dot{\mathbf{x}}$ is of the same order as evaluating ϕ_i itself, i.e. there exists a constant $c > 0$ such that for each elementary function ϕ_i*

$$\mathrm{WORK}(\phi_i' \cdot \dot{\mathbf{x}}) \leq c\ \mathrm{WORK}(\phi_i) \tag{3.12}$$

is valid (which is obviously true for the functions listed in Table 3.3), then also the effort for evaluating $\dot{\mathbf{y}}$ is of the same order as evaluating $\mathbf{y}$, i.e.

$$\mathrm{WORK}(\dot{\mathbf{y}}) \leq c\ \mathrm{WORK}(\mathbf{y}). \tag{3.13}$$

For assembling the complete Jacobian of $\mathbf{f}$ n forward propagations are needed, independent of the dimension of the image space. This makes the forward mode very attractive for calculating Jacobians of functions with low dimensional definition space and high dimensional image space. We will see in the next section, that for the reverse mode the computational complexity of calculating a gradient is proportional to the dimension of the image space and not the definition space, which is advantageous for many optimization problems.

Let us now return to our example of Section 3.2. In order to illustrate the presented approach the evaluation procedure for the directional derivative into the direction $(\dot{x}_1, \dot{x}_2)$ can be found in Algorithm 3.4.

Algorithm 3.4 Evaluation trace of the directional derivative for a simple model example

/* *Assignment phase* */
$v_{-1} = x_1 = 1.5000$
$\dot{v}_{-1} = \dot{x}_1 = 1.0000$
$v_0 = x_2 = 0.5000$
$\dot{v}_0 = \dot{x}_2 = 0.0000$
/* *Calculation phase* */
$v_1 = v_{-1}/v_0 = 1.5000/0.5000 = 3.0000$
$\dot{v}_1 = (\dot{v}_{-1} - v_{-1} \cdot \dot{v}_0)/v_0 = 1.0000/0.5000 = 2.0000$
$v_2 = \sin(v_1) = \sin(3.0000) = 0.1411$
$\dot{v}_2 = \cos(v_1) \cdot \dot{v}_1 = -0.9900 \cdot 2.0000 = -1.9800$
$v_3 = \exp(v_0) = \exp(0.5000) = 1.6487$
$\dot{v}_3 = v_3 \cdot \dot{v}_0 = 1.6487 \cdot 0.0000 = 0.0000$
$v_4 = v_1 - v_3 = 3.000 - 1.6487 = 1.3513$
$\dot{v}_4 = \dot{v}_1 - \dot{v}_3 = 2.0000 - 0.0000 = 2.0000$
$v_5 = v_2 + v_4 = 0.1411 + 1.3513 = 1.4924$
$\dot{v}_5 = \dot{v}_2 + \dot{v}_4 = -1.9800 + 2.0000 = 0.0200$
$v_6 = v_5 \cdot v_4 = 1.4924 \cdot 1.3513 = 2.0167$
$\dot{v}_6 = \dot{v}_5 \cdot v_4 + v_5 \cdot \dot{v}_4 = 0.0200 \cdot 1.3513 + 1.4924 \cdot 2.0000 = 3.0118$
/* *Assignment phase* */
$y = v_6 = 2.0167$
$\dot{y} = \dot{v}_6 = 3.0118$

3.4 The reverse mode – Propagation of gradients

This section deals with the calculation of gradients. As in Section 3.3 it is assumed that the function to be differentiated can be decomposed into a finite sequence of elementary functions being continuously differentiable.

Let us consider a function

$$\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m, \quad \mathbf{x} \mapsto \mathbf{f}(\mathbf{x}). \tag{3.14}$$

and a weighting vector $\bar{\mathbf{y}} \in \mathbb{R}^m$ and define a scalar valued function z by

$$z : \mathbb{R}^n \to \mathbb{R}, \quad \mathbf{x} \mapsto \langle \bar{\mathbf{y}}, \mathbf{f}(\mathbf{x}) \rangle_{\mathbb{R}^m} = \bar{\mathbf{y}}^T \mathbf{f}(\mathbf{x}). \tag{3.15}$$

According to the chain rule the gradient of z is defined by

$$\bar{\mathbf{x}} = \operatorname{grad} z = \operatorname{grad}\left(\bar{\mathbf{y}}^T \mathbf{f}(\mathbf{x})\right) = \bar{\mathbf{y}}^T \mathbf{f}'(\mathbf{x}) \tag{3.16}$$

where $\mathbf{f}' \in \mathbb{R}^{m\times n}$ denotes the Jacobian of $\mathbf{f}$. Geometrically $\bar{\mathbf{y}}$ and $\bar{z}$ can be interpreted in the following sense: The set $\left\{\mathbf{y} \mid \bar{\mathbf{y}}^T\mathbf{y} \equiv c\right\}$ defines a hyperplane in the image space and due to the implicit function theorem $\left\{\mathbf{x} \mid \bar{\mathbf{y}}^T\mathbf{f}(\mathbf{x}) \equiv c\right\}$ defines a smooth hypersurface in the definition space. $\bar{\mathbf{y}}$ denotes the normal of the hyperplane in the image space, $\bar{\mathbf{x}}$ the corresponding normal onto the hypersurface in the definition space.

In analogy to the previous section let us assume that $\mathbf{f}$ can be decomposed into elementary functions, i.e.

$$\mathbf{f}(\mathbf{x}) = (\phi_l \circ \phi_{l-1} \circ \cdots \circ \phi_2 \circ \phi_1)(\mathbf{x}). \tag{3.17}$$

For the gradient of z we get

$$\bar{\mathbf{x}} = \bar{\mathbf{y}}^T \cdot \phi_l' \cdot \phi_{l-1}' \cdot \cdots \cdot \phi_2' \cdot \phi_1', \tag{3.18}$$

which can be grouped as

$$\bar{\mathbf{x}} = \left(\left(\cdots\left(\left(\bar{\mathbf{y}}^T \cdot \phi_l'\right) \cdot \phi_{l-1}'\right)\cdots\right) \cdot \phi_2'\right) \cdot \phi_1' \tag{3.19}$$

due to associativity. In the frame of Section 3.2 the evaluation procedure for (3.19) can be written as shown in Algorithm 3.5. In a computer program the information to build

$$\sum_{i \succ j} \bar{v}_j \frac{\partial}{\partial v_j} \phi_i\left((v_k)_{k\prec i}\right) \tag{3.20}$$

is not available. For each ϕ_i it is known which arguments it depends on, but not which functions ϕ_i depend on a given v_j. That is why one usually avoids to form the sum over $i \succ j$ and uses an incremental setup instead. This can be found in Algorithm 3.6. It can be seen that first all intermediate variables are calculated using a normal function evaluation. Then all the adjoint quantities are calculated by executing the according statements in reverse order. The values of the intermediate variables are needed for the evaluation of the derivatives of the elementary functions and can usually not be calculated on the fly as in the forward mode. That is why they have to be saved during the forward run for the evaluation of the derivative.

One interpretation of the adjoint variables which also explains their name, is via Lagrangian multipliers. Let us view the evaluation of z in the following way: We define a constrained optimization problem

$$\bar{\mathbf{y}}^T \mathbf{y} \to \min \tag{3.21}$$

subject to the following equality constraints:

$$\phi_i\left((v_k)_{k\prec i}\right) - v_i = 0 \qquad \forall i = 1, \ldots, l \tag{3.22}$$

$$v_{l-i} - y_{l-i} = 0 \qquad \forall i = 1, \ldots, l. \tag{3.23}$$

Then, the Lagrangian variables corresponding to the equality constraints are exactly the adjoint variables computed by the reverse mode.

In Table 3.4 the adjoint operations for some elementary functions can be found. We

Algorithm 3.5 General evaluation procedure of the gradient of a function

```
/* Function evaluation to calculate values of intermediate variables */
/* Assignment phase */
for i = 1 to n do
    v_{i-n} = x_i
end for
/* Calculation phase */
for i = 1 to l do
    v_i = φ_i((v_k)_{k≺i})
end for
/* Assignment phase */
for i = m-1 to 0 step -1 do
    y_{m-i} = v_{l-i}
end for

/* Reverse sweep to calculate adjoint variables */
/* Assignment phase */
for i = 0 to m-1 do
    v̄_{l-i} = ȳ_{m-i}
end for
/* Calculation phase */
for j = l-m to 1-n step -1 do
    v̄_j = Σ_{i≻j} v̄_i ∂/∂v_j φ_i((v_k)_{k≺i})
end for
/* Assignment phase */
for i = n to 1 step -1 do
    x̄_i = v̄_{i-n}
end for
```

ϕ	$\bar{\phi}$
$w = c$	$\bar{w} = 0$
$w = u \pm v$	$\bar{u} = \bar{u} + \bar{w}$ $\bar{v} = \bar{v} + \bar{w}$
$w = u \cdot v$	$\bar{u} = \bar{u} + \bar{w} \cdot v$ $\bar{v} = \bar{v} + \bar{w} \cdot u$
$w = 1/u$	$\bar{u} = \bar{u} - \bar{w} \cdot w \cdot w$
$w = \sqrt{u}$	$\bar{u} = \bar{u} + 0.5 \cdot \bar{w}/w$
$w = u^c$	$\bar{u} = \bar{u} + (c \cdot \bar{w}) \cdot w/u$
$w = \exp(u)$	$\bar{u} = \bar{u} + \bar{w} \cdot w$
$w = \log(u)$	$\bar{u} = \bar{u} + \bar{w}/u$
$w = \sin(u)$	$\bar{u} = \bar{u} + \bar{w} \cdot \cos(u)$

Table 3.4: Elementary functions and their reverse statements for an incremental setup

Algorithm 3.6 General evaluation procedure of the gradient of a function, incremental setup

/* *Function evaluation to calculate values of intermediate variables* */
/* *Assignment phase* */
for $i = 1$ to n **do**
  $v_{i-n} = x_i$
end for
/* *Calculation phase* */
for $i = 1$ to l **do**
  $v_i = \phi_i\big((v_k)_{k \prec i}\big)$
end for
/* *Assignment phase* */
for $i = m - 1$ to 0 step -1 **do**
  $y_{m-i} = v_{l-i}$
end for

/* *Reverse sweep to calculate adjoint variables* */
/* *Assignment phase* */
for $i = 0$ to $m - 1$ **do**
  $\bar{v}_{l-i} = \bar{y}_{m-i}$
end for
for $i = 1$ to $l - m$ **do**
  $\bar{v}_i = 0$
end for
/* *Calculation phase* */
for $i = l$ to 1 step -1 **do**
  for all $j \prec i$ **do**
    $\bar{v}_j = \bar{v}_j + \bar{v}_i \frac{\partial}{\partial v_j}\phi_i\big((v_k)_{k \prec i}\big)$
  end for
end for
/* *Assignment phase* */
for $i = n$ to 1 step -1 **do**
  $\bar{x}_i = \bar{v}_{i-n}$
end for

presented an incremental setup in the table because in practice only incremental setups are used.

Up to now we always assumed no overwriting of variables, i.e. two variables never share the same storage location. However, this can be justified only for the theoretical motivation. In real programs usually several variables v_j share one common memory location. As the values of the intermediate variables are needed in the reverse sweep, one has to enhance the evaluation procedure found in Algorithm 3.6 to cope with overwriting. One way is to add *load* and *store* commands which store the values on an external file (often called tape) before being overwritten and load them from the file before being used in the reverse sweep. The enhanced version of the evaluation procedure of Algorithm 3.6 can be found in Algorithm 3.7. For a more details see GRIEWANK [67].

Coming back to Algorithm 3.6 wee see easily that for the computation of $\bar{y}$ each elementary function is processed exactly once. This implies the following for the computational complexity of a gradient evaluation.

Theorem 3.2. *Under the assumption that for each ϕ_i the effort for the reverse operation $\bar{\phi}_i$ is of the same order as evaluating the elementary function itself, i.e. there exists a constant $c > 0$ such that for each elementary function ϕ_i*

$$\text{WORK}(\bar{\phi}_i) \leq c \;\; \text{WORK}(\phi_i) \tag{3.24}$$

is valid (which is obviously true for the functions listed in Table 3.4), then the effort for evaluating $\bar{\mathbf{x}}$ is of the same order as calculating z, i.e.

$$\text{WORK}(\bar{\mathbf{x}}) \leq c \;\; \text{WORK}(z) \tag{3.25}$$

where the constant is small.

For assembling the whole Jacobian of $\mathbf{f}$ m reverse propagations are needed, independently of the number of design parameters. This makes the reverse mode very attractive for many real life problems in large scale optimization, where the image space is usually of rather low dimension whereas the definition space is high dimensional.

Let us now return to our model example of Section 3.2. In order to illustrate the presented approach the evaluation procedure for the gradient can be found in Algorithm 3.8.

3.5 Tools

Up to now we concentrated mainly on the theoretical background of the forward and the reverse mode and showed how directional derivatives and gradient information can be obtained by transforming the original computer program into a new one. But if the transformed code would have to be implemented by hand, this would not only be time consuming but also pretty error-prone. Furthermore, there would be only few advantages to completely hand-coded routines (the most important one is that it is clear which part of the derivative code has to be changed when the code for the function evaluation changes). That is why in the AD-community tools were developed to support this program transformation during the last decade. Nevertheless it is necessary to understand the fundamentals of AD in order to apply these tools efficiently.

Algorithm 3.7 General evaluation procedure of the gradient of a function, incremental setup with overwriting of variables

/* *Function evaluation to calculate values of intermediate variables* */
/* *Assignment phase* */
for $i = 1$ to n **do**
  $v_{i-n} = x_i$
end for
/* *Calculation phase* */
for $i = 1$ to l **do**
  STORE v_i
  $v_i = \phi_i\big((v_k)_{k \prec i}\big)$
end for
/* *Assignment phase* */
for $i = m - 1$ to 0 step -1 **do**
  $y_{m-i} = v_{l-i}$
end for

/* *Reverse sweep to calculate adjoint variables* */
/* *Assignment phase* */
for $i = 0$ to $m - 1$ **do**
  $\bar{v}_{l-i} = \bar{y}_{m-i}$
end for
for $i = 1$ to $l - m$ **do**
  $\bar{v}_i = 0$
end for
/* *Calculation phase* */
for $i = l$ to 1 step -1 **do**
  LOAD v_i
  for all $j \prec i$ **do**
    $\bar{v}_j = \bar{v}_j + \bar{v}_i \frac{\partial}{\partial v_j} \phi_i\big((v_k)_{k \prec i}\big)$
  end for
  $\bar{v}_i = 0$
end for
/* *Assignment phase* */
for $i = n$ to 1 step -1 **do**
  $\bar{x}_i = \bar{v}_{i-n}$
end for

Algorithm 3.8 Evaluation trace of the gradient for a simple model example

/* *Function evaluation to calculate values of intermediate variables* */
/* *Assignment phase* */
$v_{-1} = x_1 = 1.5000$
$v_0 = x_2 = 0.5000$
/* *Calculation phase* */
$v_1 = v_{-1}/v_0 = 1.5000/0.5000 = 3.0000$
$v_2 = \sin(v_1) = \sin(3.0000) = 0.1411$
$v_3 = \exp(v_0) = \exp(0.5000) = 1.6487$
$v_4 = v_1 - v_3 = 3.000 - 1.6487 = 1.3513$
$v_5 = v_2 + v_4 = 0.1411 + 1.3513 = 1.4924$
$v_6 = v_5 \cdot v_4 = 1.4924 \cdot 1.3513 = 2.0167$
/* *Assignment phase* */
$y = v_6 = 2.0167$

/* *Reverse sweep to calculate adjoint variables* */
/* *Assignment phase* */
$\bar{v}_6 = \bar{y} = 1.0000$
$\bar{v}_0 = \bar{v}_1 = \bar{v}_2 = \bar{v}_3 = \bar{v}_4 = \bar{v}_5 = 0.0000$
/* *Calculation phase* */
$\bar{v}_5 = \bar{v}_5 + \bar{y} \cdot v_4 = 0.0000 + 1.0000 \cdot 1.3513 = 1.3513$
$\bar{v}_4 = \bar{v}_4 + \bar{y} \cdot v_5 = 0.0000 + 1.0000 \cdot 1.4924 = 1.4924$
$\bar{v}_2 = \bar{v}_2 + \bar{v}_5 = 1.3513$
$\bar{v}_4 = \bar{v}_4 + \bar{v}_5 = 1.4924 + 1.3513 = 2.8437$
$\bar{v}_3 = \bar{v}_3 - \bar{v}_4 = -2.8437$
$\bar{v}_1 = \bar{v}_1 + \bar{v}_4 = 2.8437$
$\bar{v}_0 = \bar{v}_0 + \bar{v}_3 \cdot v_3 = 0.0000 - 2.8437 \cdot 1.6487 = -4.6884$
$\bar{v}_1 = \bar{v}_1 + \bar{v}_2 \cdot \cos(v_1) = 2.8437 + 1.3514 \cdot (-0.9900) = 1.5059$
$\bar{v}_0 = \bar{v}_0 - \bar{v}_1 \cdot v_1/v_0 = -4.6884 - 1.5059 \cdot 3.0000/0.5000 = -13.7239$
$\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1/v_0 = 0.0000 + 1.5059/0.5000 = 3.0118$
/* *Assignment phase* */
$\bar{x}_2 = \bar{v}_0 = -13.7239$
$\bar{x}_1 = \bar{v}_{-1} = 3.0118$

$$\begin{tikzcd}
\text{func.src} \arrow[r, "\textsc{Source-to-source transformation}"] \arrow[d, "\textsc{Augmentation}"] & \text{func\&der.src} \arrow[d, "\textsc{Compilation}"] \\
\text{func++.src} \arrow[r, "\textsc{Compilation}"'] & \text{func\&der.obj}
\end{tikzcd}$$

Figure 3.2: From function sources to object files containing derivative information

The two basic methodologies used by these tools are operator overloading and source-to-source transformation. Even though these concepts look very different, both can be brought into a common frame which can be found in Figure 3.2.

Both approaches start from an implementation of the function evaluation. When applying tools based on operator overloading (e.g. ADOL-C [69], FADBAD [14]) this source has to be augmented by hand in order to define the independent and the dependent variables as well as the intermediate ones. Then, this augmented function evaluation is compiled using a standard compiler.

The situation is slightly different, when using tools based on source-to-source transformation tools (e.g. ADIFOR [18], TAF [150], or ADIC [20]). Here the source of the function evaluation is analyzed using a tool which works similar to a compiler. It analyzes the function evaluation to set up a dependence graph. For this analysis the user has to specify the independent and the dependent variables. Using the dependence graph these tools generate source code of the derivatives as their output which can be compiled using a standard compiler. In both cases the object files together with libraries delivered by the tool implementers contain all the information needed for evaluating derivatives. In the following we want to discuss this process using two of the tools, TAF and ADOL-C as typical examples for source-to-source translators and tools based on operator overloading, respectively.

3.5.1 The source-to-source translator TAF

TAF, which stands short for *Transformation of Algorithms in Fortran* is a source-to-source translator that generates Fortran routines out of Fortran routines. It is a commercial product developed by Giering and Kaminsky at FastOpt (`http://www.fastopt.de`) and is the successor of TAMC, the *Tangent linear and Adjoint Model Compiler* (GIERING [58], GIERING AND KAMINSKI [61]) which is available under `http://puddle.mit.edu/~ralf/tamc/`. The routines to be differentiated have to be implemented in Fortran-77 or Fortran-90, only operator overloading is not supported. Derivatives are computed in forward mode using a tangent-linear model or in reverse mode using an adjoint model. Additionally, a code for products between the Jacobian and a vector or matrix can be generated.

```
double precision function f (x1, x2)
double precision x1, x2
double precision q, d, dummy, mysin, myexp

q = x1 / x2
d = q - myexp(x2)
dummy = q * x2

f = (mysin(q) + d) * d

return
end


double precision function mysin(x)
double precision x

mysin = dsin(x)
return
end


double precision function myexp(x)
double precision x

myexp = dexp(x)
return
end
```

Figure 3.3: Fortran-77 code for a simple model problem

Given the independent and dependent variables of the top-level routine (all these quantities are specified using command line parameters) TAF applies an inter-procedural data dependence analysis and an inter-procedural data flow analysis to determine all code parts which have to be differentiated and all intermediate variables for which derivatives have to be calculated.

The principles of source-to-source transformation can be found in GIERING AND KAMINSKY [59, 60]. They describe rules for deriving the adjoint statements needed which can also be used for manual adjoint derivation. These rules form the basis for the algorithms implemented in TAMC and TAF.

An implementation of our simple model problem in Fortran-77 as well as its derivatives via forward and reverse mode can be found in Figure 3.3, Figure 3.4, Figure 3.5 and Figure 3.6 respectively. The code for the derivatives was generated using TAMC with the commands

```
tamc -toplevel f -input "x1,x2" -forward func.f
```

```
      subroutine g_f( x1, x2, f, g_x1, g_x2, g_g )
C**************************************************************
C** This routine was generated by the                        **
C** Tangent linear and Adjoint Model Compiler,  TAMC 5.3.2    **
C**************************************************************
      double precision f, x1, x2, g_g, g_x1, g_x2
      double precision d, dh, fh, q, g_d, g_dh, g_fh, g_q, myexp, mysin
C----------------------------------------------
C TANGENT LINEAR AND FUNCTION STATEMENTS
C----------------------------------------------
      g_q = g_x1/x2-g_x2*(x1/(x2*x2))
      q = x1/x2
      call g_hmyexp( x2,dh,g_x2,g_dh )
      g_d = (-g_dh)+g_q
      d = q-dh
      call g_hmysin( q,fh,g_q,g_fh )
      g_g = g_d*(fh+2*d)+g_fh*d
      f = (fh+d)*d

      end


      subroutine g_hmyexp( x, myexp, g_x, g_myexp )
      double precision g_myexp, g_x, myexp, x
C----------------------------------------------
C TANGENT LINEAR AND FUNCTION STATEMENTS
C----------------------------------------------
      g_myexp = g_x*dexp(x)
      myexp = dexp(x)

      end


      subroutine g_hmysin( x, mysin, g_x, g_mysin )
      double precision g_mysin, g_x, mysin, x
C----------------------------------------------
C TANGENT LINEAR AND FUNCTION STATEMENTS
C----------------------------------------------
      g_mysin = g_x*dcos(x)
      mysin = dsin(x)

      end
```

Figure 3.4: Fortran-77 code for the directional derivative, generated by TAMC V 5.3.2

```
      subroutine adf( x1, x2, f, adx1, adx2, adg )
C**************************************************************
C** This routine was generated by the                        **
C** Tangent linear and Adjoint Model Compiler,  TAMC 5.3.2    **
C**************************************************************
      double precision adg, adx1, adx2, f, x1, x2
      double precision add, addi, adfi, adq, d, q, myexp, mysin
C----------------------------------------------
C RESET LOCAL ADJOINT VARIABLES
C----------------------------------------------
      add = 0.d0
      adq = 0.d0

C----------------------------------------------
C ROUTINE BODY
C----------------------------------------------
C----------------------------------------------
C FUNCTION AND TAPE COMPUTATIONS
C----------------------------------------------
      q = x1/x2
      d = q-myexp(x2)
      f = (mysin(q)+d)*d

C----------------------------------------------
C ADJOINT COMPUTATIONS
C----------------------------------------------
      add = add+adg*(2*d+mysin(q))
      adfi = adg*d
      call adhmysin( q,adq,adfi )
      adfi = 0.d0
      adg = 0.d0
      addi = -add
      adq = adq+add
      call adhmyexp( x2,adx2,addi )
      addi = 0.d0
      add = 0.d0
      adx1 = adx1+adq/x2
      adx2 = adx2-adq*(x1/(x2*x2))
      adq = 0.d0


      end
```

Figure 3.5: Fortran-77 code for the gradient, generated by TAMC V 5.3.2, Part 1

```
      subroutine adhmyexp( x, adx, admyexp )
C**************************************************************
C** This routine was generated by the                        **
C** Tangent linear and Adjoint Model Compiler,  TAMC 5.3.2    **
C**************************************************************
      double precision admyexp, adx, x

C----------------------------------------------
C ROUTINE BODY
C----------------------------------------------
      adx = adx+admyexp*dexp(x)
      admyexp = 0.d0

      end


      subroutine adhmysin( x, adx, admysin )
C**************************************************************
C** This routine was generated by the                        **
C** Tangent linear and Adjoint Model Compiler,  TAMC 5.3.2    **
C**************************************************************
      double precision admysin, adx, x

C----------------------------------------------
C ROUTINE BODY
C----------------------------------------------
      adx = adx+admysin*dcos(x)
      admysin = 0.d0

      end
```

Figure 3.6: Fortran-77 code for the gradient, generated by TAMC V 5.3.2, Part 2

and

```
tamc -toplevel f -input "x1,x2" -reverse func.f,
```

respectively. In the implementation of the function evaluation we added a dummy variable, which depends on the input variables but does not influence the output quantities. When the function evaluation is compiled, an optimizing compiler detects this fact and does not generate code for this statement. TAMC applies similar data flow and data dependence analysis and also detected this fact. That is why it does not generate any derivative code for this unnecessary statement.

3.5.2 The operator overloading package ADOL-C

ADOL-C (Griewank et al. [70, 69]) which stands short for *Automatic differentiation by overloading in C++* is one example for AD-packages based on operator overloading. It is developed by the group of Griewank at the Technical University of Dresden and is available free of charge under `http://www.math.tu-dresden.de/wir/project/adolc/index.html`. The routines to be differentiated have to be implemented in C or C++, the augmented code is C++ code. The package facilitates the evaluation of first and higher order derivatives. The resulting derivative evaluation routines may be called from C/C++, Fortran, or any other language that can be linked with C. For scalar-valued functions ADOL-C provides easy to use drivers for function and gradient evaluation as well as Hessian times vector products and Hessian evaluation. For vector valued function besides function and Jacobian evaluation routines also products between a vector and the Jacobian are available.

In order to get derivative information the code must be augmented in the following sense: First the active region containing the function evaluation has to be specified. The key ingredient of all AD packages using operator overloading is the concept of *active variables*. Within the active region, all variables depending directly or indirectly on the independent variables have to be replaced by active variables. In ADOL-C this is realized by replacing the corresponding `double` variables by variables of the type `adouble`. For these variables derivative information is generated. To complete the augmentation of the code, the dependent and the independent variables have to be specified. Special care has to be taken on the taping of conditional statements, but we do not want go into details here (see Griewank et al. [69]).

During the evaluation of the active region the overloaded operations for the type `adouble` record all the operations within the active region where active variables are involved on a so called *tape*. This tape is used by an interpreter to evaluate the function, gradients, etc. These evaluations only use the tape, no more the augmented function. That is why it is possible to generate a tape using one program and to use the function defined by the tape for example as objective of an optimization within another.

In order to illustrate this let us look at our simple model example. An implementation in C can be found in Figure 3.7. It is more or less a transcript of the Fortran implementation in Figure 3.3. For brevity we removed all necessary preprocessor statements and pre-declarations needed to get the code fragment to compile. Figure 3.8 contains the augmented code. We assumed that `f` contains the top-level function to be differentiated. It can be seen that the two code fragments differ only sightly in the evaluation itself. The main difference is that all intermediate variables were changed from `double` to `adouble`. Additionally in the top-level function the dependent and independent variables as well as the active region have to be specified.

```
double f (double x1, double x2)
  {
    double q, d, value;

    q = x1 / x2;
    d = q - myexp(x2);

    value = (mysin(q) + d) * d;

    return value;
  }


double mysin(double x)
  {
    double value;

    value = sin(x);
    return value;
  }


double myexp(double x)
  {
     double value;

     value = exp(x);
     return value;
  }
```

Figure 3.7: C code for a simple model problem

```
double f (double dx1, double dx2)
  {
    trace_on(1234);               // the active region starts here

    adouble x1, x2;

    x1 <<= dx1;                  // define x1 and x2 as
    x2 <<= dx2;                  // independent variables

    // define all intermediate variables as active
    adouble q, d, value;

    q = x1 / x2;
    d = q - myexp(x2);

    value = (mysin(q) + d) * d;

    double dummy;
    value >>= dummy;            // define dependent variables

    trace_off(1234);            // the active region ends here

    return dummy;
  }


adouble mysin(adouble x)
  {
    adouble value;

    value = sin(x);
    return value;
  }


adouble myexp(adouble x)
  {
    adouble value;

    value = exp(x);
    return value;
  }
```

Figure 3.8: Augmented C++ code for a simple model problem

```
void main ()
  {
    double x[2], f, grad[2];

    x[0] = 1.5;               // initialize the parameter
    x[1] = 0.5;


    // evaluate the function
    function (1234, 2, x, &f);

    // evaluate the gradient
    gradient (1234, 2, x, grad);
  }
```

Figure 3.9: Function and gradient evaluation for a simple model example in ADOL-C

When the tape is generated it can be used as a replacement for the function evaluation itself as the evaluation drivers only take the information on the tape. A code fragment in C for evaluating the function and its gradient can be found in Figure 3.9.

Chapter 4

A Black-Box Strategy Using an Elimination Approach

4.1 Introduction

Optimal design problems are often divided, accordingly to their number of design parameters, into problems with few design parameters (maximal dimension of the design parameter vector $\mathbf{q}$ is approximately 100) and into problems with very large design spaces (e.g. $\dim(\mathbf{q}) \approx \dim(\mathbf{u})$). In this chapter we want to present a strategy suitable for rather few design parameters only. The objective and constraints can be more or less arbitrary, but smooth. We restrict ourselves to elliptic state problems, but remark on the changes for time-dependent state problems where necessary. The next chapter will deal with the situation of having many design parameters. The method presented there differs considerably, although common aspects exist which will be discussed there.

In order to illustrate the method we use a model example – the minimization of the mass of the frame of an injection moulding machine under constraints on the deformation and stresses. This real life example shall also demonstrate the applicability of the method in an industrial design process. There, mainly two goals have to be fulfilled:

- On the one hand, tools are needed which are flexible enough to handle the various requirements. Nevertheless, they have to be robust to produce reliable results. Especially it is desirable to spend only a little work on modifying the code, when the requirements change.

- On the other hand, these tools have to be fast. Up to now, design changes are often made by engineers by hand following mainly their experience and intuition. Unfortunately, due to lack of time, this process has to be stopped after a few iterations, in most cases only two or three. Then, the best design obtained so far, is taken. Fast tools automatizing some parts of the design process strongly accelerate the design process itself, and by far more design drafts can be considered.

This chapter will begin with the introduction and modeling of our model problem, followed by the corresponding optimal design problem. By using a solution operator for the state problem this optimal design problem can be transformed into an equivalent optimization problem which is easier manageable by the optimizer. Additionally, extensions to the

Figure 4.1: Cross section of the original shape

general SQP-frame presented in Section 2.2 are introduced which remove the demand for second order derivatives of the Lagrangian. The main part of this chapter is devoted to gradient and sensitivity calculation. Since implementing analytic derivatives is rather error prone and an improper approach for an industrial design process, various alternatives are presented, analyzed, and compared to each other. Besides finite differences, the direct and the adjoint sensitivity method, several approaches using automatic differentiation are presented. A numerical comparison of some of the presented approaches concludes this chapter.

4.2 Model example: Optimal sizing of a machine frame

The frame of an injection moulding machine is briefly sketched by its 2D-cut Ω given in Figure 4.1. For a frame of homogeneous thickness, typical dimensions are:

- thickness of one plate: 180 mm
- mass of one plate: 3.8 tons
- clumping force (surface force): 300 tons $\approx$ 16 N/mm^2
- length: 2.8 m
- height: 1.7 m
- 2 supporting areas

The primary goal of the design phase is to minimize the mass of the frame of the injection moulding machine. Several other requirements have to be fulfilled in addition, e.g.

- maximal v. Mises stress: $\sigma^{\text{vM}} \le \sigma^{\text{vM}}_{\max}$,
- maximal tensile stress: $\sigma^{\text{ten}} \le \sigma^{\text{ten}}_{\max}$,

- shrinking angle of the clumping unit (vertical edges on top, called *wings*): $\alpha \leq \alpha_{\max}$,
- handling of the machine or the feeding mechanism,
- easy and cost efficient manufacturing.

For the definition of the v. Mises stress and the tensile stress see e.g. ZIEGLER [162].

Some of these constraints can be integrated into the optimization procedure directly (e.g. restrictions on the stresses), whereas others like the easy manufacturing have to be considered in a post-processing step.

In order to evaluate the stresses σ, the displacement field u of the frame under some load F has to be known.

For a fixed thickness $q(x) \in Q$, the displacement field $u(x), x = (x_1, x_2) \in \Omega$, fulfills

$$a(q; u, v) = F(v), \qquad \forall\, v \in U, \tag{4.1}$$

with

$$a(q; u, v) = \int_\Omega q \, \frac{\partial u_i}{\partial x_j} \, E_{ijkl} \, \frac{\partial v_k}{\partial x_l} \, \mathrm{d}x, \quad F(v) = \int_{\Gamma_N} g \, v \, \mathrm{d}s$$

where E_{ijkl} denotes the elasticity tensor and g the surface force density on a part Γ_N of the boundary. Volume forces are neglected. u and v are assumed to be in

$$U = \left\{ v \in [H^1(\Omega)]^2 \mid v = 0 \text{ on } \Gamma_D, \text{meas } \Gamma_D > 0 \right\} \tag{4.2}$$

(set of admissible displacements) where $\partial\Omega = \Gamma_D \cup \Gamma_N$, $\Gamma_D = \overline{\Gamma}_D$ and $\Gamma_D \cap \Gamma_N = \emptyset$.

E_{ijkl} is given by

$$E_{ijkl} = \lambda \delta_{ij} \delta_{kl} + \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}), \tag{4.3}$$

where δ_{ij} denotes the Kronecker-Delta and λ, μ denote Lame's constants. These can be calculated from Young's modulus E and Poisson's ratio ν by

$$\lambda = \frac{E\,\nu}{(1+\nu)(1-\nu)}, \qquad \mu = \frac{E}{2(1+\nu)}. \tag{4.4}$$

In (4.1) we have assumed that

- a generalized plane stress problem is considered, i.e. we consider a body Ω as a plate that is thin in x_3-direction (compared to the other coordinate directions) and that can carry stresses only parallel to the x_1-x_2-plane, and
- the applied surface tractions are independent of x_3, i.e.

  $$g(x) = (g_1(x), g_2(x)),$$

  and therefore, there is no displacement in x_3-direction and the other two displacement components are also independent of x_3.

The design problem can be stated as follows:

$$\begin{aligned}
&\int_\Omega q(x)\ \mathrm{d}x \to \min_{(u,q)\in U\times Q} \\
\text{subject to}\quad & a(q;u,v) = F(v), && \forall\ v \in U, \\
& \sigma^{\mathrm{vM}}(u) \le \sigma^{\mathrm{vM}}_{\max} && \text{a.e. in } \Omega, \\
& \sigma^{\mathrm{ten}}(u) \le \sigma^{\mathrm{ten}}_{\max} && \text{a.e. in } \Omega, \\
& \alpha(u) \le \alpha_{\max}, \\
& 0 < \underline{q} \le q \le \overline{q}, && \text{a.e. in } \Omega.
\end{aligned} \tag{4.5}$$

$\sigma^{\mathrm{vM}}(u)$ denotes the v. Mises stress, $\sigma^{\mathrm{ten}}(u)$ the tensile stress in the frame, $\underline{q}, \overline{q} \in \mathbb{R}^+$. The change in the shrinking angle of the clumping unit (vertical edges on top, called *wings*) is denoted by $\alpha(u)$.

For discretizing the problem, we use triangular finite elements with piece-wise constant shape functions for approximating q and piece-wise quadratic ones for approximating u. In many situations it is additionally assumed that q is constant in certain non-overlapping sub-regions Ω_i. We denote the discrete approximation of q and u by q_h and u_h where $q_h \in Q_h$ and $u_h \in U_h$.

Summarizing all those considerations, the discretized optimization problem can be formulated as follows:

$$\begin{aligned}
&\int_{\Omega_h} q_h(x)\,\mathrm{d}x \to \min_{(u_h,q_h)\in U_h\times Q_h} \\
& a(q_h;u_h,v_h) = F(v_h), && \forall\ v_h \in U_h, \\
& \sigma^{\mathrm{vM}}(u_h) \le \sigma^{\mathrm{vM}}_{\max}, \\
& \sigma^{\mathrm{ten}}(u_h) \le \sigma^{\mathrm{ten}}_{\max}, \\
& \alpha(u_h) \le \alpha_{\max}, \\
& 0 < \underline{q} \le q_h \le \overline{q},
\end{aligned} \tag{4.6}$$

where Ω_h denotes the discretized domain.

Choosing bases

$$\Phi = (\phi_1, \ldots, \phi_m)^T \in U_h \qquad \text{and} \qquad \Sigma = (\sigma_1, \ldots, \sigma_n)^T \in Q_h \tag{4.7}$$

of the finite dimensional spaces U_h and Q_h we may represent $(u_h, q_h) \in U_h \times Q_h$ via

$$u_h = \mathbf{u}^T\Phi, \qquad q_h = \mathbf{q}^T\Sigma \tag{4.8}$$

with coordinate vectors $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{q} \in \mathbb{R}^n$. This allows us to rewrite (4.6) in matrix-vector form as

$$\begin{aligned}
&\mathbf{m}^T\,\mathbf{q} \to \min_{(\mathbf{u},\mathbf{q})\in\mathbb{R}^m\times\mathbb{R}^n} \\
&\mathbf{K}(\mathbf{q})\,\mathbf{u} = \mathbf{F} \\
&\sigma^{\mathrm{vM}}(\mathbf{u}) \le \sigma^{\mathrm{vM}}_{\max} \\
&\sigma^{\mathrm{ten}}(\mathbf{u}) \le \sigma^{\mathrm{ten}}_{\max} \\
&\alpha(\mathbf{u}) \le \alpha_{\max} \\
&0 < \underline{q} \le \mathbf{q} \le \overline{q},
\end{aligned} \tag{4.9}$$

with the symmetric positive definite, large, sparse stiffness matrix $\mathbf{K}$ and the vectors $\mathbf{m}$ and $\mathbf{F}$. The constraints on σ^{vM}, σ^{ten} and $\mathbf{q}$ have to be understood component-wise.

In our application, the upper limits on the angle and the stresses are either treated as constraints or as soft limits, which can be violated to some extent, if the mass would be severely smaller then. Furthermore, the pointwise constraints on σ^{vM} and σ^{ten} are replaced by using a higher order ℓ^p norm

$$\|\mathbf{v}\|_p = \sqrt[p]{\sum |v_i|^p}.$$

Treating the upper limits as soft constraints leads to the following reformulation:

$$\begin{aligned}
\mathbf{m}^T\,\mathbf{q} + \omega_1\big(\max\left(\|\sigma^{\mathrm{vM}}(\mathbf{u})\|_p - \sigma^{\mathrm{vM}}_{\max}, 0\right)\big)^2 & \\
+\, \omega_2\big(\max\left(\|\sigma^{\mathrm{ten}}(\mathbf{u})\|_p - \sigma^{\mathrm{ten}}_{\max}, 0\right)\big)^2 & \\
+\, \omega_3\big(\max\left(\alpha(\mathbf{u}) - \alpha_{\max}, 0\right)\big)^2 & \quad\to \min_{(\mathbf{u},\mathbf{q})\in\mathbb{R}^m\times\mathbb{R}^n} \\
\mathbf{K}(\mathbf{q})\,\mathbf{u} = \mathbf{F} & \\
0 < \underline{q} \le \mathbf{q} \le \overline{q} &
\end{aligned} \tag{4.10}$$

where $\omega_i, i = 1,\ldots,3$ denote user-chosen factors of influence. Note that this modification leads to an objective in C^1. It looks similar to a penalty formulation of the constraints, but the weights $\omega_i, i = 1,\ldots,3$ are kept fixed during the iteration and do not tend to infinity like in penalty methods. Therefore, the problem of ill-conditioning of the Hessian is avoided.

Usually, the state equation in (4.10) is not fulfilled exactly as iterative solvers are used. Then, it is necessary to adopt the convergence criterion of the iterative solver to the discretization parameter h. As long as the difference between the exact discrete solution and the approximate solution calculated by the iterative solver is of the same order as the approximation of the continuous solution by the used discretization scheme, we do not have to pay special attention to the iterative solver. The exact discrete solution and the approximate solution are both approximations of the solution of the continuous problem of the same order. From now on, we will always assume this, and therefore, we will not distinguish between the exact discrete solution and the one calculated by an iterative solver.

4.3 A black-box strategy for optimal design

From the optimization's point of view the problem (4.10) is a special case of

$$\begin{aligned}
& J(\mathbf{u},\mathbf{q}) \to \min_{(\mathbf{u},\mathbf{q})\in\mathbb{R}^m\times\mathbb{R}^n} \\
\text{subject to}\quad & \mathbf{K}(\mathbf{q})\,\mathbf{u} = \mathbf{F} \\
& \underline{q} \le \mathbf{q} \le \overline{q},
\end{aligned} \tag{4.11}$$

where $\mathbf{q}$ denotes the vector of design parameters and $\mathbf{u}$ the solution of the governing finite element (FE) state equation. The splitting of the parameter vector into design parameters $\mathbf{q}$ and the solution of the state equation $\mathbf{u}$ is typical for problems for optimal design. From the optimization's point of view the discretized state equation can be interpreted as equality constraints. For our model example it is linear with respect to $\mathbf{u}$ and $\mathbf{K}(\mathbf{q})$ is symmetric and positive definite for all admissible parameters $\mathbf{q}$. We introduce a solution operator $\mathbf{S}(\mathbf{q})$ fulfilling

$$\mathbf{S} : \mathbf{q} \mapsto \mathbf{u}(\mathbf{q}) \qquad \text{with} \qquad \mathbf{K}(\mathbf{q})\,\mathbf{u}(\mathbf{q}) = \mathbf{F}(\mathbf{q}). \tag{4.12}$$

For a general nonlinear state equation $\mathbf{e}(\mathbf{u}, \mathbf{q}) = \mathbf{F}$ the solution operator can be defined in an analogous way. Also for time dependent state problems a solution operator can be introduced. We want to remark, that such an operator does not always exist (e.g. for our model problems, if q_h vanishes on several neighboring elements). But from now on, we always assume the existence of $\mathbf{S}(\mathbf{q})$ for all admissible $\mathbf{q}$.

Using this solution operator, we can eliminate $\mathbf{u}$ formally which leads to

$$\begin{aligned} & \tilde{J}(\mathbf{q}) = J(\mathbf{S}(\mathbf{q}), \mathbf{q}) \to \min_{\mathbf{q} \in \mathbb{R}^n} \\ \text{subject to} \qquad & \underline{q} \leq \mathbf{q} \leq \overline{q}. \end{aligned} \tag{4.13}$$

Since we want to use a standard SQP method for optimizing the problem, the formulation (4.13) is advantageous compared to (4.11) as it has much fewer parameters. This relies on the fact, that standard implementations of the SQP method are based on linear algebra with dense matrices.

On the contrary to the presentation in Section 2.2 standard SQP methods do not use the Hessian of the Lagrangian. They replace it by approximate Hessian information, e.g. using a Quasi-Newton approximation formula. Our code uses a modified BFGS update formula following POWELL [122] in order to avoid the need for second derivatives of the objective. The quadratic subproblem is solved by a range space based QP method (c.f. GILL, MURRAY, AND WRIGHT [63]) combined with an active index set strategy. Other alternatives would be e.g. null-space based QP methods or the dual method of GOLDFARB AND IDNANI [65] which is used for instance in the code of SCHITTKOWSKI [137]. The main advantage of the latter compared to the other two is that it does not need a feasible start point.

We use a line search procedure for globalizing our SQP method, which uses an exact penalty function

$$\Phi_k(\mathbf{q}) = \tilde{J}(\mathbf{q}) + \sum_{j=1}^{n} \overline{\sigma}_j^k \max(q_j - \overline{q}, 0) + \underline{\sigma}_j^k \max(\underline{q} - q_j, 0) \tag{4.14}$$

with suitable chosen penalty parameters $\overline{\sigma}_j^k$, $\underline{\sigma}_j^k$ as merit function (c.f. HAN [78]). As usual, q_i denotes the i-th component of $\mathbf{q}$ in (4.14), k denotes the iteration index.

A short sketch onto a model SQP algorithm for solving

$$\begin{aligned} & \tilde{J}(\mathbf{q}) \to \min_{\mathbf{q} \in \mathbb{R}^n} \\ \text{subject to} \qquad & \mathbf{c}(\mathbf{q}) \leq 0 \end{aligned} \tag{4.15}$$

is given in Algorithm 4.1.

4.4 Calculating gradients

Using a Quasi-Newton strategy and update formulas within the SQP method as proposed in the section before, the remaining main problem is the calculation of gradients for the objective and the constraints. As the problem was transformed into a problem with box constraints

Algorithm 4.1 SQP model algorithm

Require: A suitable starting point $\mathbf{q_0}$

$\mathbf{B}_0 = \mathbf{I}$
$\mathbf{g}_0 = \mathrm{grad}\,\tilde{J}(\mathbf{q}_0)$

/ linearize constraints,* $\doteq$ *denotes a first order approximation */*
$\mathbf{c}(\mathbf{q}_0 + \boldsymbol{\delta}) \doteq \mathbf{A}_0\,\boldsymbol{\delta} + \boldsymbol{b}_0$
$k = 0$

while not converged **do**
/ Calculate search direction* $\mathbf{s}_k$ **/*
Solve
$\frac{1}{2}\mathbf{s}^T\,\mathbf{B}_k\,\mathbf{s} + \mathbf{g}_k^T\,\mathbf{s} \to \min_{\mathbf{s}}$
under the constraints $\mathbf{A}_k\,\mathbf{s} \leq -\mathbf{b}_k - \mathbf{A}_k\,\mathbf{q}_k$

/ Line search procedure */*
Calculate $\alpha_k \in (0, 1]$ as large as possible such that

$\Phi_k(\mathbf{q}_k) + \mu_1\alpha_k\,\Phi_k'(\mathbf{q}_k) \leq \Phi_k(\mathbf{q}_k + \alpha_k\,\mathbf{s}_k) \leq \Phi_k(\mathbf{q}_k) + \mu_2\alpha_k\,\Phi_k'(\mathbf{q}_k)$
with $0 < \mu_1 < \frac{1}{2} < \mu_2 < 1$ with a suitable merit function Φ_k

/ Update several quantities */*
$\mathbf{q}_{k+1} = \mathbf{q}_k + \alpha_k\,\mathbf{s}_k$
$\mathbf{g}_{k+1} = \mathrm{grad}\,\tilde{J}(\mathbf{q}_{k+1})$
$\mathbf{c}(\mathbf{q}_{k+1} + \boldsymbol{\delta}) \doteq \mathbf{A}_{k+1}\,\boldsymbol{\delta} + \mathbf{b}_{k+1}$
Update Hessian approximation $\to \mathbf{B}_{k+1}$
$k = k + 1$
end while

only, routines providing analytic gradients for these constraints can be implemented easily. But for the objective, the implementation of an analytic derivative is by far too complicated and time consuming. Furthermore, it would not be well suited for the use in a design process, as we would loose the flexibility of the code completely. That is the reason why we have to think of alternative methods for calculating the gradients.

Several methods are presented and compared to each other in this section. On the one hand we have black box methods like finite differences or automatic differentiation (c.f. GRIEWANK [67]), on the other hand, methods exploiting the special structure of the state equation are available, e.g. the direct method or the adjoint method (c.f. HASLINGER AND NEITTAANMÄKI [81]).

As none of these methods is well suited for an industrial design process, a hybrid method combining automatic differentiation and the adjoint method is developed.

4.4.1 Finite differences

If no analytic derivatives of a function can be implemented due to the high complexity then an approximation by finite differences is often the first idea. One approximation is the central difference quotient

$$\mathrm{D}_h f(x) = \frac{f(x+h) - f(x-h)}{2h}. \tag{4.16}$$

The choice of the increment h is rather critical for getting accurate results and depends on estimates of the third derivative of f.

In order to improve the accuracy of finite differences, extrapolation methods can be used. For an initial increment H the sequence

$$\mathrm{D}_H f, \mathrm{D}_{\frac{H}{2}} f, \mathrm{D}_{\frac{H}{4}} f, \ldots, \mathrm{D}_{\frac{H}{2^i}} f, \ldots \tag{4.17}$$

is calculated and extrapolated for $i \to \infty$. These methods return not only a value for the derivative, but also an estimate of the accuracy of that value which can be used for controlling the order of the extrapolation scheme (c.f. STOER [143] or DEUFLHARD AND HOHMANN [46]).

The main properties are summarized as follows:

- Two function evaluations are needed per difference quotient and in most cases several difference quotients are needed in order to reach the desired accuracy of the derivative. Furthermore, the number of function evaluations is proportional to the number of design parameters. Due to the high effort finite differences are only well suited for problems with few design parameters.

- Finite differences can easily be used for very complex functions, as they do not rely on any special properties. On the other hand they can not exploit any special properties of the function which makes the use of finite differences rather inefficient in certain cases.

- From the user's point of view finite differences are very flexible. Changes in the desired objective imply only a re-implementation of the function calculating the objective. Time consuming changes of the gradient routine do not appear, which is especially important for the acceptance of such a method in an industrial design process.

- The possible use of iterative methods for solving the state equation is very important for our problem as the number of parameters in the state equation may be rather large.

As finite differences do not rely on any special properties of the function the coupling with iterative solvers can be done without any problems.

4.4.2 Automatic differentiation

Compared to finite differences, AD follows a completely different approach. Finite differences try to approximate the derivative and therefore do not provide accurate results, whereas AD methods incur no truncation error at all and usually yield derivatives with working accuracy. Starting point is a computer program that calculates numerical values for a function. First, a symbolic evaluation graph mapping the design parameters to the function values is built. Like symbolic differentiation, AD operates by systematic application of the chain rule, familiar from elementary differential calculus. However, in the case of AD, the chain rule is applied not to symbolic expressions, but to numerical values. By using all the intermediate results generated by the function evaluation, the exponential growth of the evaluation complexity of symbolic differentiation can be avoided, as many common subexpressions can be used repeatedly when the gradient is evaluated at any particular point. Furthermore, optimizations made for the function evaluation also pay off for its derivative. Details on how the AD technique works, as well as many related issues on calculating higher order derivatives can be found in GRIEWANK [67]. A short introduction into the methodology of AD is presented in Chapter 3.

Two different kinds of tools are known in the AD community: The first group is based on source-to-source transformation, e.g. ADIFOR (c.f. BISCHOF, CARLE, KHADEMI, AND MAUER [19]) written for FORTRAN codes , of TAF (`http://www.fastopt.de`) written also for FORTRAN codes. The other group is based on evaluation graphs generated at runtime, e.g. ADOL-C written for C and C++ codes (c.f. GRIEWANK, JUEDES, AND UTKE [70]). As our finite element code is completely written in C++ and uses heavily virtual inheritance, source code transformation tools can not be used. It has to be mentioned, that some of the properties of AD listed in the following rely on the use of runtime tools.

- ADOL-C needs a file containing the evaluation graph in a symbolic form for evaluating the function and its gradient. This file is generated at runtime. For optimal design problems, huge memory and disk capabilities are required for that purpose. Due to the need of an evaluation graph, ADOL-C can only be applied to functions of moderate complexity. The limiting factor is not the inherent complexity of the function itself, but the size of the generated files and the time needed for reading and writing the data. To give an example, the files storing the evaluation graph for our model problem discretized with about 450 design parameters and about 7500 DOFs in the FE state equation needs about 1 GB of disk space.

- The flexibility of AD with respect to changes in the objective is similar to finite differences. Changes in the objective need only a re-implementation of the objective function, but no changes in the gradient routine when runtime tools are used. Sometimes, special care has to be taken for a correct generation of the evaluation graph, especially in the context of conditional statements.

- AD using ADOL-C is a black box method. The use of the evaluation graph is a drawback of the method, especially when debugging is needed. This is compensated to some extent by the good runtime behavior of the method. For the reverse mode (c.f. Section 3.4), the calculation time of the gradient is independent of the number of design parameters and takes the time of about 15 – 20 native C++ function evaluations as long as the evaluation graph can be stored in the main memory of the computer. Compared to the use of finite differences, this is a tremendous speedup, even for problems with only 10 – 20 design parameters.

- The coupling of AD with iterative solvers is a problem of current research (see e.g. GRIEWANK [67] and references therein). Since the use of iterative methods (e.g. multilevel methods) is important for solving fine discretizations of the state equation efficiently, the applicability is limited to problems, where direct solvers can be used.

4.4.3 Direct and adjoint method

The direct and the adjoint method are both well-known in the shape optimization community (see e.g. HASLINGER AND NEITTAANMÄKI [81]) and take into account the special structure of the state equation. They differentiate the state equation with respect to a design parameter q_i. For our model example, this leads to

$$\mathbf{K}\frac{\partial \mathbf{u}}{\partial q_i} = \frac{\partial \mathbf{F}}{\partial q_i} - \frac{\partial \mathbf{K}}{\partial q_i}\mathbf{u}. \tag{4.18}$$

For the direct method, (4.18) is solved numerically using the same methods as for the state problem itself. Then the gradient of the objective can be calculated by

$$\frac{\mathrm{d}\tilde{J}}{\mathrm{d}q_i} = \frac{\partial J}{\partial q_i} + \langle \frac{\partial J}{\partial \mathbf{u}}, \frac{\partial \mathbf{u}}{\partial q_i} \rangle. \tag{4.19}$$

On the contrary to the direct method, the adjoint method solves (4.18) formally and inserts the result in (4.19) which leads to

$$\frac{\mathrm{d}\tilde{J}}{\mathrm{d}q_i} = \frac{\partial J}{\partial q_i} + \langle \mathbf{K}^{-T}\frac{\partial J}{\partial \mathbf{u}}, \frac{\partial \mathbf{F}}{\partial q_i} - \frac{\partial \mathbf{K}}{\partial q_i}\mathbf{u} \rangle \tag{4.20}$$

For a general state equation

$$\mathbf{e}(\mathbf{u}, \mathbf{q}) = \mathbf{F}$$

the situation is slightly more difficult. Here, we introduce a solution operator for the linearized state problem (linearized with respect to $\mathbf{u}$) of the form

$$\mathbf{S}_{\mathrm{lin}}(\mathbf{q}) = \left(\frac{\partial \mathbf{e}}{\partial \mathbf{u}}\right)^{-1}. \tag{4.21}$$

For the direct method we get the representation

$$\frac{\partial \mathbf{u}}{\partial q_i} = \mathbf{S}_{\mathrm{lin}} \cdot \left(\frac{\partial \mathbf{F}}{\partial q_i} - \frac{\partial \mathbf{e}}{\partial q_i}\right) \tag{4.22}$$

and calculate the gradient of the objective using (4.19). For the adjoint method, we insert the formal representation of $\frac{\partial \mathbf{u}}{\partial q_i}$ into (4.19) which leads to

$$\frac{\mathrm{d}\tilde{J}}{\mathrm{d}q_i} = \frac{\partial J}{\partial q_i} + \langle \mathbf{S}_{\mathrm{lin}}^T\frac{\partial J}{\partial \mathbf{u}}, \frac{\partial \mathbf{F}}{\partial q_i} - \frac{\partial \mathbf{e}}{\partial q_i} \rangle \tag{4.23}$$

instead of (4.20).

We want to mention that for time dependent state problems $\mathbf{S}_{\text{lin}}$ represents a forward integration of the linearized state equation in time with given initial values, whereas $\mathbf{S}_{\text{lin}}^T$ represents a backward integration of the linearized state problem with given final values. If the state equation is non-linear with respect to $\mathbf{u}$ the adjoint method needs the complete evaluation trace of the forward integration to evaluate $\mathbf{S}_{\text{lin}}^T$. This usually results in huge memory requirements for storing the values of the forward run. For an alternative using a check-pointing strategy, i.e. storing only a view intermediate values and recalculating the remaining ones, see e.g. Griewank [66], Griewank and Walther [71], or Charpentier [36].

In the following the main properties are summarized:

- The direct method needs one solution of the state equation per design parameter, whereas the adjoint method needs the solution of one adjoint problem for the objective and in principle for each constraint. As $\mathbf{K}$ is symmetric in our case, the effort for solving the adjoint problem is the same as for solving the state equation itself. Depending on the number of design parameter and constraints, the better suited method can be chosen.

- As analytic partial derivatives of J with respect to $\mathbf{q}$ and $\mathbf{u}$ are needed ($\frac{\partial J}{\partial \mathbf{q}}$, $\frac{\partial J}{\partial \mathbf{u}}$, $\frac{\partial \mathbf{K}}{\partial \mathbf{q}}$, $\frac{\partial \mathbf{F}}{\partial \mathbf{q}}$), both methods can only be applied to simple objectives, where this can easily be done. Furthermore, the flexibility of the method suffers from the need of hand-coded gradient routines.

- Compared to finite differences or the use of AD for the whole function, this approach is much faster. Finite differences need much more solutions of the design problem, compared to AD the huge evaluation graph which originates mainly from the solution of the state equation is avoided.

- Any solver can be used for solving the state problem, especially the use of iterative solvers like conjugated gradient methods with multilevel preconditioning is appropriate.

4.4.4 Hybrid method

Comparing the methods presented in the last two sections, it can be seen that the strengths of these methods lie in completely different areas. AD provides very high flexibility with respect to the used objective, but has severe drawbacks with respect to memory requirements of the used computer, the use of iterative solvers for the state equation and with respect to longer runtime. In contrast, the direct and the adjoint method can be combined easily with iterative solvers and provide a fast way for calculating the needed gradients, but they lack from the needed flexibility. Hence, we combine both approaches into a new hybrid method.

The main drawback of the direct or the adjoint method is the need of analytic partial derivatives of the objective and the constraints with respect to $\mathbf{q}$ and $\mathbf{u}$. But these derivatives can easily be provided by using AD. Then, only $\frac{\partial \mathbf{K}}{\partial \mathbf{q}}$ and $\frac{\partial \mathbf{F}}{\partial \mathbf{q}}$ remain, for which hand-coded routines have to be implemented or AD can be used. For optimal sizing problems, these routines can be hand-coded easily. Furthermore, they do not depend on the specific problem which justifies the additional effort for coding even for complex state problems.

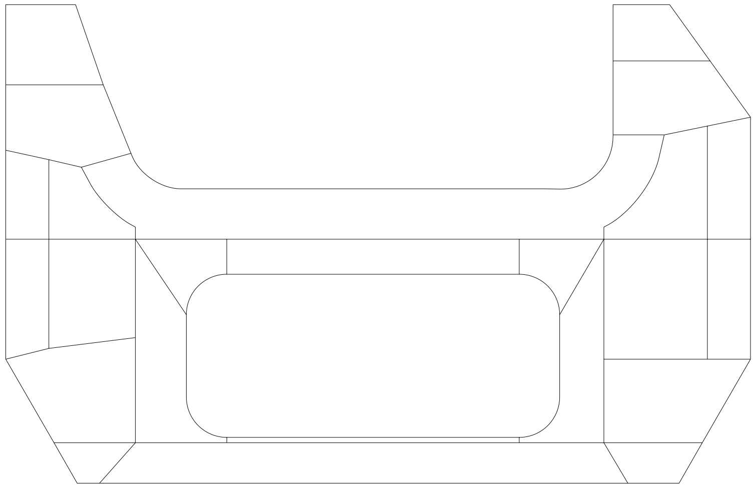

Figure 4.2: Frame with 24 sub-domains

		Finite Diff.	Pure AD	Hybrid M.
Problem dim.	Nr. of design params	24	24	24
	Nr. of elements (quad.)	3 981	3 981	3 981
	DOFs of state equ.	16 690	16 690	16 690
Optimizer statistics	Iterations	83	100	100
	Function evaluations	19 752	315	236
	Gradient evaluations	84	101	101
Runtime	Total CPU time	12.4 h	4.88 h	0.39 h
	Total elapsed time	12.6 h	8.42 h	0.40 h
Elapsed time	Optimizer	0.01 h	0.03 h	0.01 h
	Function evaluation	0.23 h	2.36 h	0.20 h
	Gradient evaluation	12.40 h	6.00 h	0.18 h

Table 4.1: Comparison of the runtime for various differentiation strategies

4.5 Numerical results

In the following, some numerical results for the problem stated in Section 4.2 are presented. They were calculated on an SGI Origin 2000 with 300 MHz.

At the beginning, we tried to use only a few design parameters. Therefore, we divided our domain into a number of sub-domains (see Figure 4.2) and approximated the thickness with a constant function in each sub-domain. The state equation was discretized using triangular finite elements with quadratic FE functions.

For evaluating the gradient, either finite differences, a pure AD approach or the hybrid method were used. For a better comparison, the calculation was terminated after a fixed number of steps (the run using finite differences terminated earlier because the search direction was no descent direction anymore). Detailed results can be found in Table 4.1. All three methods lead to a similar design with about 5 % reduction of the mass compared to the

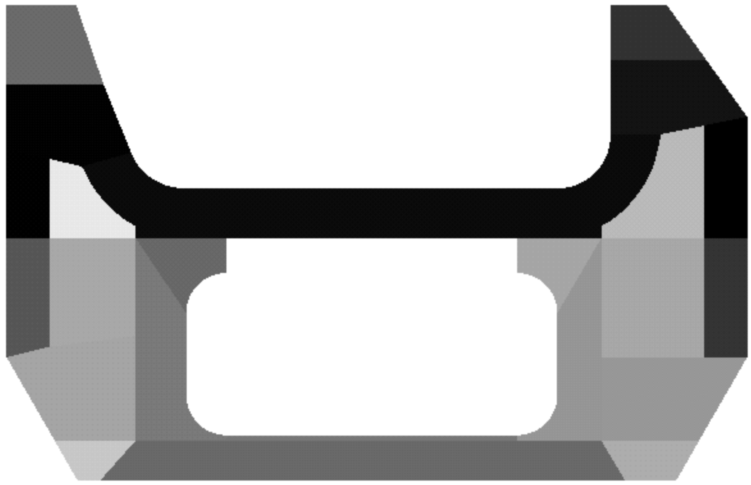

Figure 4.3: Optimized thickness distribution for 24 sub-domains

starting configuration (which is the current design of the frame). The optimized thickness distribution can be seen in Figure 4.3 (the darker the color, the thicker the frame), Figure 4.4 shows the distribution of the van Mises stresses in the optimized frame (the lighter the color, the higher the stresses).

It can be seen in Table 4.1 that for a few design parameters the main effort consists in solving the FE state equation, respectively calculating the gradient of the objective. Compared to finite differences and the pure AD approach, the hybrid method is much faster, as it combines a fast function evaluation and a fast gradient evaluation. The gradient evaluation is the main drawback of finite differences. For the pure AD approach we had to implement additional safeguards. In order to detect when a regeneration of the evaluation graph was necessary, we compared the value of the objective using the evaluation graph with the value using a native C++ implementation which explains the longer runtime of the function evaluation.

In order to find a better suited splitting of the domain in few sub-domains we increased the number of sub-domains to about 450. We used the coarsest grid of our FE triangulation also for discretizing the thickness distribution (c.f. Figure 4.5). For solving the design problem, each coarse grid finite element was subdivided into 16 elements using 2 levels of uniform refinement. On this refined triangulation the state equation was discretized using finite elements with quadratic FE functions. As finite differences are no more suitable for this number of design parameters, Table 4.2 contains only results for the pure AD approach and the hybrid method.

The values reported for the evaluation graph have the following meaning: *Independents* is the number of independent variables of the function differentiated automatically, *dependents* the number of dependent ones (there is only one dependent variable, as only the objective is differentiated using ADOL-C). *Operations* gives us the number of arithmetic operations in the evaluation graph, *maxlive* the maximal number of active variables (maximal number of variables allocated at one point of time during the evaluation of the objective), *valstacksize* the number of intermediate results which have to be stored in AD's reverse mode.

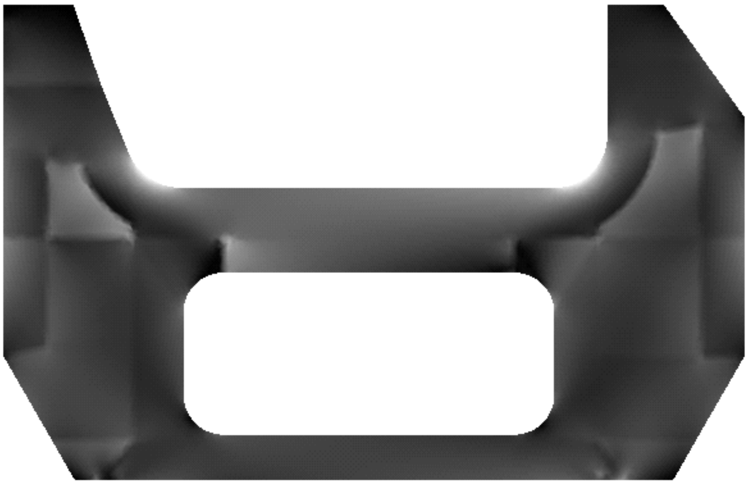

Figure 4.4: Van Mises stresses for 24 sub-domains

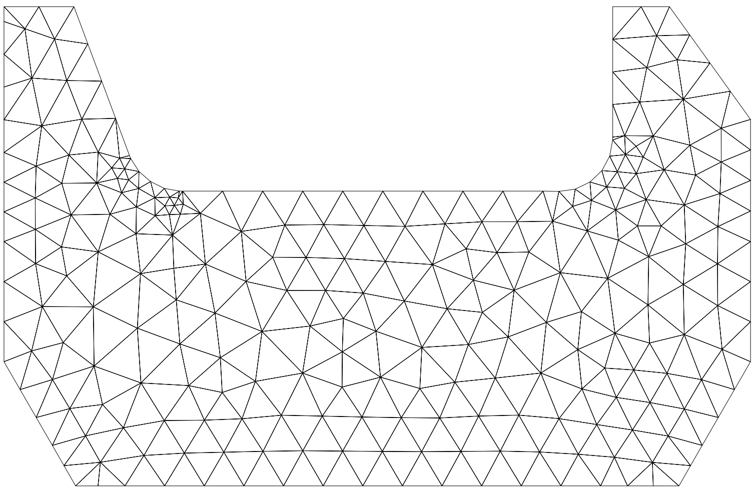

Figure 4.5: Frame with 449 sub-domains

		Pure AD	Hybrid M.
Problem dim.	Nr. of design params	449	449
	Nr. of elements (quad.)	1 796	1 796
	DOFs of state equ.	7 518	7 518
Evaluation graph	Independents	449	9 314
	Dependents	1	1
	Operations	45 521 797	1 399 910
	Maxlive	540 302	28 140
	Valstacksize	51 995 116	1 644 461
	Total file size	953 MB	32.4 MB
Optimizer Statistics	Iterations	800	800
	Function evaluations	5 811	3 744
	Gradient evaluations	801	801
Runtime	Total CPU time	32.3 h	3.73 h
	Total runtime	38.5 h	3.76 h
Elapsed time	Optimizer	4.0 h	1.93 h
	Function evaluation	16.5 h	1.29 h
	Gradient evaluation	18.0 h	0.54 h

Table 4.2: Comparison of the runtime for many design parameters

The optimized thickness distribution can be seen in Figure 4.6, the corresponding stress distribution in Figure 4.7. Analyzing the runtime behavior of the two methods in Table 4.2, it can be seen that the pure AD approach is no more competitive due to the large file containing the evaluation graph. Furthermore, it can be seen that for the hybrid approach the optimizer needs already a considerable amount of the total runtime.

In Table 4.3 we compared the runtime of the hybrid method for different discretizations of the state and design space. It can be seen that the relative amount of the runtime needed by the optimizer even grows when using more design parameters as the complexity of one optimization step is proportional to $(\dim \mathbf{q})^3$ (due to the use of dense matrix linear algebra), whereas the complexity of solving one FE state equation is proportional to $\dim \mathbf{u}$ (if solvers with optimal complexity e.g. the conjugate gradient method with appropriate multigrid or multilevel preconditioning are used).

For comparison to the results in Figure 4.6 and Figure 4.7, the thickness distribution was discretized using about 1100 design parameters (see Table 4.3). The optimized thickness distribution and the corresponding distribution of the v. Mises stresses can be found in Figure 4.8 and Figure 4.9 respectively. The main problem using this large number of design parameters is the runtime needed by our optimization module, which dominates the time needed by all function and gradient evaluations completely (about 90 % of the runtime, about 18000 DOFs of the FE state equation). In the next chapter we will develop a method which can also cope with even larger numbers of design parameters.

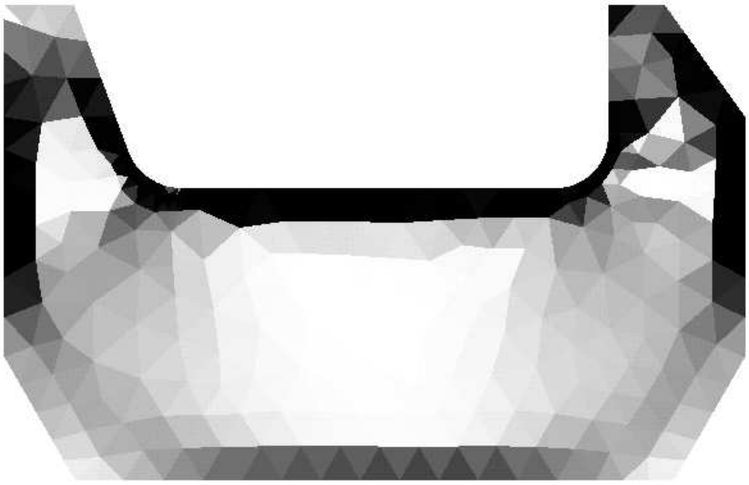

Figure 4.6: Optimized thickness distribution for 449 sub-domains

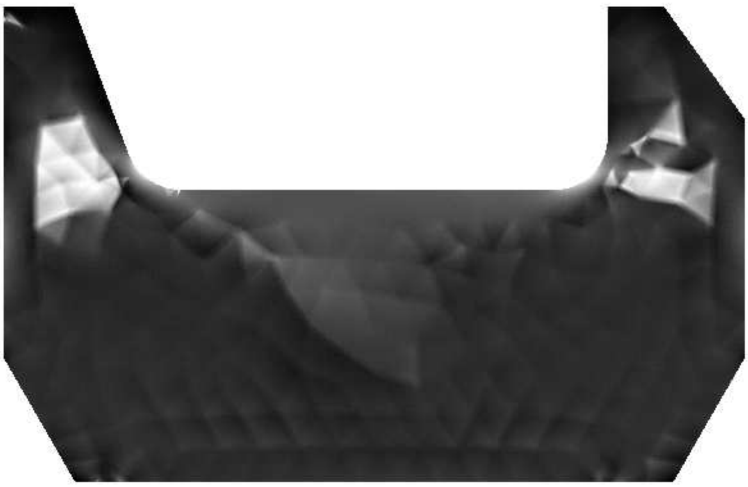

Figure 4.7: Van Mises stresses for 449 sub-domains

		Hybrid M.	Hybrid M.	Hybrid M.
General	Nr. of design params	449	449	1 078
	Nr. of elements	1 796	7 184	4 312
	DOFs of state equ.	7 518	29 402	9 028
Evaluation graph	Independents	9 314	36 586	22 368
	Operations	1 399 910	5 578 270	3 361 017
	Total file size	32.4 MB	129.2 MB	77.9 MB
Runtime	Iterations	800	800	2 200
	Total CPU time	3.73 h	14.01 h	104.0 h
	Total runtime	3.76 h	14.12 h	105.1 h
Elapsed time	Optimizer	1.93 h	2.64 h	90.3 h
	Function evaluation	1.29 h	8.13 h	9.8 h
	Gradient evaluation	0.54 h	3.35 h	3.9 h

Table 4.3: Scale up of the hybrid method in design and state space

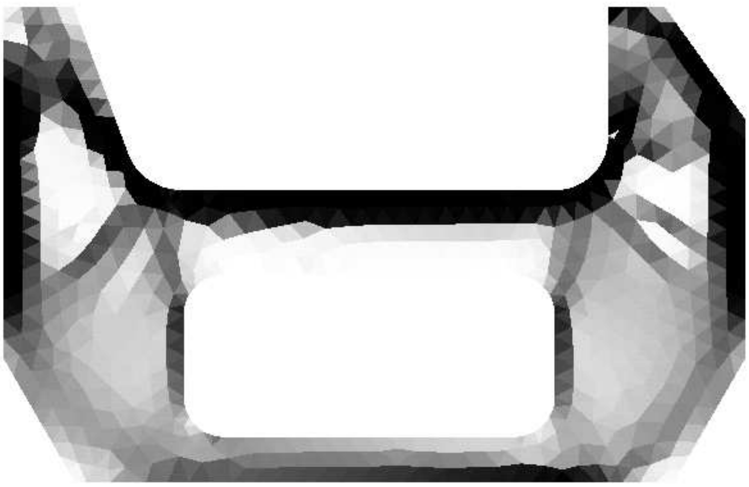

Figure 4.8: Optimized thickness distribution for 1078 sub-domains

Figure 4.9: Van Mises stresses for 1078 sub-domains

Chapter 5

An All-At-Once Approach for Large Design Spaces

5.1 Introduction

In the previous chapter we introduced a method which is very flexible due to the use of AD. On the other hand problems with respect to the runtime appear if the number of design parameters increases. As early as about 1000 design parameters the method reaches practically its limits as one optimization run takes about 4 days computing time (c.f. Table 4.3). In the current realization, increasing the number of design parameters to 2000 would let the computing time increase about by a factor of 16 resulting in 2 months computing time.

Analyzing the runtime behavior of the black-box strategy presented in Section 4.3 in more detail one observes the following:

- The solution of the state problem can not be significantly improved any more. When using a multilevel preconditioned CG method for solving our state problem the order of complexity of our solver would be optimal. We currently use a sparse direct solver as preconditioner in our CG method. Additionally, we update the preconditioner, i.e. factorize the current stiffness matrix, only when the number of CG steps exceeds a certain limit, otherwise we use the factorization of a previous stiffness matrix as preconditioner. This explains the non-optimal increase in the runtime for function and gradient evaluation in Table 4.3 as both need the solution of the state problem, respectively of the adjoint linearized state problem.

- The optimizer itself can only slightly be improved. The most time consuming part – solving the QP problem – already uses update formulas for a faster calculation of the basis of the range space when constraints are added or removed.

The main reason for the strong increase in the runtime when increasing the number of design parameters is inherent in the used optimization strategy and is twofold:

- On the one hand, by introducing a solution operator it is necessary to solve the state problem in each function evaluation, i.e. the state variable is always admissible with respect to the state equation. This calculation is rather easy for state problems linear in the state variable $\mathbf{u}$, but can be highly time consuming for nonlinear state problems.

Additionally each gradient evaluation of the objective needs one solution of a problem adjoined to the linearized state equation.

- On the other hand, the introduction of a solution operator $\mathbf{S}$ and formal elimination of the state variable as presented in Section 4.3 destroy the sparsity structure of the underlying optimal design problem in most cases. This can be explained as follows: Let us consider the optimization problem

$$\begin{aligned} & J(\mathbf{u},\mathbf{q}) \rightarrow \min_{(\mathbf{u},\mathbf{q})\in\mathbb{R}^m\times\mathbb{R}^n} \\ \text{subject to} \quad & \mathbf{e}(\mathbf{u},\mathbf{q}) = \mathbf{F} \end{aligned} \tag{5.1}$$

where $\mathbf{e}(\mathbf{u},\mathbf{q}) = \mathbf{F}$ denotes the state equation. Additionally, assume that the Hessian of the Lagrangian

$$\mathcal{L}(\mathbf{u},\mathbf{q}) = J(\mathbf{u},\mathbf{q}) + \boldsymbol{\lambda}^T\left(\mathbf{e}(\mathbf{u},\mathbf{q}) - \mathbf{F}\right) \tag{5.2}$$

and the Jacobian of the state equation $\mathbf{e}(\mathbf{u},\mathbf{q})$ are sparse. These are assumptions which are fulfilled very often for optimal design problems. When applying the strategy presented in the previous chapter, we introduce a solution operator $\mathbf{S}$ for the state equation, which is usually not sparse anymore. For the model problem treated in the previous chapter $\mathbf{S} = \mathbf{K}^{-1}(\mathbf{q})\,\mathbf{F}(\mathbf{q})$ holds. Then the Hessian of the reduced problem

$$\tilde{J}(\mathbf{q}) = J(\mathbf{S}(\mathbf{q}),\mathbf{q}) \rightarrow \min_{\mathbf{q}\in\mathbb{R}^\mathbf{n}} \tag{5.3}$$

with

$$\mathbf{e}(\mathbf{S}(\mathbf{q}),\mathbf{q}) = \mathbf{F} \tag{5.4}$$

is given by

$$\begin{aligned} \nabla^2_{\mathbf{qq}}\tilde{J} = \nabla^2_{\mathbf{qq}}\mathcal{L} - \nabla^2_{\mathbf{qu}}\mathcal{L}\ \mathbf{S}_{\text{lin}}\ \nabla_{\mathbf{q}}\mathbf{e} - \\ (\nabla_{\mathbf{q}}\mathbf{e})^T\ \mathbf{S}_{\text{lin}}^T\ \nabla^2_{\mathbf{uq}}\mathcal{L} + (\nabla_{\mathbf{q}}\mathbf{e})^T\ \mathbf{S}_{\text{lin}}^T\ \nabla^2_{\mathbf{uu}}\mathcal{L}\ \mathbf{S}_{\text{lin}}\ \nabla_{\mathbf{q}}\mathbf{e} \end{aligned} \tag{5.5}$$

where $\mathbf{S}_{\text{lin}}$ denotes the solution operator of the linearized state problem with respect to $\mathbf{u}$, and $\mathbf{S}_{\text{lin}}^T$ denotes the solution operator of its adjoint problem. As $\mathbf{S}_{\text{lin}}$ is usually not sparse (c.f. Section 4.3), the Hessian of $\tilde{J}$, the objective used in the approach of the previous section, is usually dense. This does not matter, as long as the design space is small, but is a severe problem when the design space becomes large.

As a consequence a different approach without introducing a solution operator is used. This all-at-once approach considers the optimal design problem in the product space $U \times Q$ of state space U and design space Q. The state equation is no more eliminated formally but treated as an equality constraint during the optimization.

We will begin this chapter by introducing a simple model problem in Section 5.2. This parameter identification problem has the same structure as an optimal design problem, but usually simpler objectives are used. We will continue with the presentation of the optimization procedures in Section 5.3. An analysis of the well-posedness of the occurring quadratic programming subproblems as well as an overview over the necessary preliminaries is also presented. Section 5.4 deals with the discretization, Section 5.5 and Section 5.6 with the numerical realization of the presented algorithm. As we use an iterative scheme, we describe also several ways of preconditioning the occurring equation systems. Section 5.7 presents numerical results showing the potential of this approach for problems with large design spaces. At the end of this chapter we will discuss how this method can be applied for optimal design problems, which changes are necessary and which problems occur there, as well as some approaches how to handle these problems.

5.2 Model problem: Parameter identification

Since distributed parameters have to be determined from indirect measurements in many applications that are modeled by PDEs, parameter identification has become an important part of mathematical modeling. One of the main characteristics of the majority of these problems is that they are ill-posed, i.e. the parameter does not depend on the data in a stable way. As the data can not be measured exactly in practice, regularization methods have to be used in order to obtain a stable solution in the presence of data noise. We do not want to focus here on the various aspects of regularization but refer to the literature (see e.g. ENGL, HANKE, AND NEUBAUER [50] or KIRSCH [99]). Furthermore, the analysis with respect to convergence to the exact solution when the data noise tends to 0 is excluded, we refer to BURGER AND MÜHLHUBER [33] for details on this topic. Also the convergence of the discrete approximation to the solution of the parameter identification will not be presented in this frame, for details see BURGER AND MÜHLHUBER [34].

In this context we only want to present the class of parameter identification problems. We will see that from an abstract point of view these are very similar to optimal design problems. The main difference between these two classes is that the latter has typically by far more complicated objectives. In the last section we come back to this aspect with a short discussion.

The basic setup of the identification problems treated as model examples is as follows: Given an observation

$$z = E\hat{u}, \tag{5.6}$$

where $E : U \to Z$ is a bounded linear operator and $\hat{u}$ denotes the exact state. We want to identify the parameter $q \in Q_{ad} \subset Q$ in an underlying equation

$$e(u, q) = f, \tag{5.7}$$

where $e : U \times Q \to Y$ is a continuous nonlinear operator. In this setup we assume that Q and Z are Hilbert spaces, that Q_{ad} is a closed subset of Q with nonempty interior and that U and Y are appropriate Banach spaces. In addition, we assume that the operator e is homogeneous, i.e,

$$e(0, 0) = 0, \tag{5.8}$$

which is no restriction of generality, since for an arbitrary operator e we can transform (5.7) into an equivalent equation with homogeneous operator via

$$\tilde{e}(u, q) := e(u, q) - e(0, 0), \qquad \tilde{f} = f - e(0, 0).$$

In practice one has to deal with data z^δ that are corrupted by noise instead of the exact data z. We assume that the observation error is bounded by

$$\|z - z^\delta\|_Z \leq \delta, \tag{5.9}$$

where $z = E\hat{u}$ such that there exists a $\hat{q} \in Q_{ad}$ with

$$e(\hat{u}, \hat{q}) = f. \tag{5.10}$$

The pair $(\hat{u}, \hat{q})$ denotes an exact solution of the parameter identification problem (as introduced above).

Note that under typical conditions, parameter identification problems in equations of the form (5.6), (5.7) are ill-posed, i.e., an arbitrarily small error in the data z can lead to an arbitrarily large deviation in the reconstructed parameter q. In presence of noise, a solution of the equation $Eu = z^\delta$ does not always exist, and therefore one has to consider the corresponding normal equation respectively the least-squares problem

$$\frac{1}{2}\|Eu - z^\delta\|^2 \to \min_{(u,q)\in U\times Q} \tag{5.11}$$

subject to the equality constraint (5.7) and to $q \in Q_{ad}$. Since our main focus is the treatment of the state equation as equality constraint, we will omit the additional constraint $q \in Q_{ad}$ in the following. This results in the problem

$$\begin{aligned} \frac{1}{2}\|Eu - z^\delta\|^2 &\to \min_{(u,q)\in U\times Q} \\ \text{subject to} \qquad e(u,q) &= f \end{aligned} \tag{5.12}$$

which will be used as model problem.

5.3 Optimization procedures in the product space

One possibility to solve our model problem is to apply an SQP method as presented in Section 2.2 or Algorithm 4.1 to (5.12). As explained in the introduction of this chapter, an SQP method in the product space $U \times Q$ has advantages compared to the introduction of a solution operator of the state problem, when the dimension of the discretized design space is large. We will introduce the SQP method here in a function space frame and will consider the influence of the discretization later.

5.3.1 SQP methods in the product space

Applying a standard SQP scheme to our model example (5.12) leads to the following method:

Method 1 (Sequential Quadratic Programming Method in Product Space). *Let $(u_0, q_0, \lambda_0) \in U \times Q \times Y^*$ be a given initial value. The method of* product space sequential quadratic programming *(PSSQP) consists of the iteration procedure*

$$(u_{k+1}, q_{k+1}, \lambda_{k+1}) = (\overline{u}_k, \overline{q}_k, \overline{\lambda}_k), \tag{5.13}$$

where $(\overline{u}_k, \overline{q}_k)$ is the minimizer of the quadratic programming problem

$$\frac{1}{2}\|Eu - z^\delta\|_Z^2 + \frac{1}{2}\langle \lambda_k, e''(u_k, q_k)(u - u_k, q - q_k)^2\rangle \to \min_{(u,q)\in U\times Q}, \tag{5.14}$$

with $\langle\cdot,\cdot\rangle$ denoting the usual duality product on $Y^* \times Y$. By $e'(u,q)(v,s)$ and $e''(u,q)(v,s)^2$ we denote the first and second directional derivatives of e in direction (v,s) evaluated at (u,q). The minimization is subject to the linear constraint

$$e(u_k, q_k) + e'(u_k, q_k)(u - u_k, q - q_k) = f, \tag{5.15}$$

where $\overline{\lambda}_k$ is the corresponding Lagrange-multiplier.

Because of the ill-posedness of our model problem, a direct application of standard SQP-type methods as presented in Method 1 is not possible, since a minimizer of the quadratic problems arising in each of the iteration steps needs not exist and if one exists, it might not depend on the data in a stable way. Therefore we will modify our SQP-type approach, which leads to stable quadratic subproblems due to an additional penalty term in the parameter space. In Burger and Mühlhuber [33] the following iterative regularization scheme was proposed, which is a modification of Method 1:

Method 2 (Iteratively Regularized Sequential Quadratic Programming Method). *Let $(u_0, q_0, \lambda_0) \in U \times Q \times Y^*$ be a given initial value and let $(\beta_k)_{k\in\mathbb{N}}$ be a bounded sequence of positive real numbers. The method of* iteratively regularized sequential quadratic programming *(IRSQP) consists of the iteration procedure*

$$(u_{k+1}, q_{k+1}, \lambda_{k+1}) = (\overline{u}_k, \overline{q}_k, \overline{\lambda}_k), \tag{5.16}$$

where $(\overline{u}_k, \overline{q}_k)$ is the minimizer of the quadratic programming problem

$$\frac{1}{2}\|Eu - z^\delta\|_Z^2 + \frac{\beta_k}{2}\|q - q_k\|_Q^2 + \frac{1}{2}\langle\lambda_k, e''(u_k, q_k)(u - u_k, q - q_k)^2\rangle \to \min_{(u,q)\in U\times Q}. \tag{5.17}$$

The minimization is subject to the linear constraint

$$e(u_k, q_k) + e'(u_k, q_k)(u - u_k, q - q_k) = f, \tag{5.18}$$

where $\overline{\lambda}_k$ is the corresponding Lagrange-multiplier.

The IRSQP-method involves second order derivatives of the operator e. As we are interested in the case of attainable data here, the Lagrangian variable must be small close to a solution. That is why, these second order derivatives are usually ignored for least square problems. Exploiting this fact, we introduce a variant of Method 2, which takes into account the special structure of the objective functional.

Method 3 (Levenberg-Marquardt Sequential Quadratic Programming Method). *Let $(u_0, q_0) \in U \times Q$ be a given initial value and let $(\beta_k)_{k\in\mathbb{N}}$ be a bounded sequence of positive real numbers. The* Levenberg-Marquardt sequential quadratic programming *method (LM-SQP) consists of the iteration procedure*

$$(u_{k+1}, q_{k+1}) = (\overline{u}_k, \overline{q}_k), \tag{5.19}$$

where $(\overline{u}_k, \overline{q}_k) \in U \times Q$ is the minimizer of the quadratic programming problem

$$\frac{1}{2}\|Eu - z^\delta\|_Z^2 + \frac{\beta_k}{2}\|q - q_k\|_Q^2 \to \min_{(u,q)\in U\times Q}, \tag{5.20}$$

subject to the linear constraint

$$e(u_k, q_k) + e'(u_k, q_k)(u - u_k, q - q_k) = f. \tag{5.21}$$

An important issue of applying the above stated methods is the question, whether the iteration procedure is well-defined. Besides the existence and uniqueness of minimizers of the quadratic programming problems, the stable dependence of the iterates on the previous iterates and on the data is of high interest. We will investigate these questions in the following subsection.

5.3.2 Well-posedness of the quadratic programming problems

In the following we will verify the well-posedness of the quadratic programming problem (5.20), (5.21) under reasonable assumptions on the state equation e. Besides that, we will also analyze the KKT-System of the problem in the frame of linear saddle point problems. Later in this chapter this approach will also be used for solving the occurring quadratic programming problems.

In typical applications, the equation (5.7), respectively its linearization, admits a unique solution with respect to the state, i.e.,

$$e_u(u,q)^{-1} : Y \to U \text{ exists and is a continuous linear operator for all } (u,q) \in U \times Q. \tag{5.22}$$

Under this assumption we can prove the well-posedness of Method 3 which is done in the following proposition:

Proposition 5.1. *Let e be continuously Fréchet-differentiable, let* (5.22) *hold and let $\beta_k > 0$. Then the quadratic programming problem* (5.20), (5.21) *has a unique solution $(\overline{u}_k, \overline{q}_k) \in U \times Q$, which is also the only local minimum.*

Proof. The basis of the proof is that the admissible set of the linearized state equation is closed, convex and non-empty. Furthermore the objective is strictly convex. Using the main theorem of convex optimization we may conclude that there exists a unique global minimum and no further local minima. Details can be found in BURGER AND MÜHLHUBER [33]. □

In a similar way we can show that under certain restrictions on the regularization parameter, the IRSQP method is well-defined and that the quadratic programming problem (5.17), (5.18) has a unique solution which is also the only local minimum. For the details of the proof, as well as the exact assumptions see BURGER AND MÜHLHUBER [33].

As already explained in the previous section, the well-posedness can usually not be shown for Method 1. Due to the ill-posedness of our model problem, the quadratic programming problem (5.14), (5.15) need not have a unique solution. Even if a unique minimizer exists, it need not depend on the data or on the previous iterate in a stable way.

So far we have not discussed the Lagrangian of the problem and the arising first-order optimality conditions. These are not only necessary but also sufficient under the assumptions needed for showing the well-posedness of the quadratic programming problems, since the objective functionals are strictly convex.

5.3.3 The Karush-Kuhn-Tucker system

Based on the standard theory of convex optimization, we can formulate the Lagrangian of the problems (5.17), (5.18) and (5.20), (5.21) as

$$\begin{aligned}\mathcal{L}_k(u,q;\lambda) = {} & \frac{1}{2}\|Eu - z^\delta\|_Z^2 + \frac{\beta_k}{2}\|q - q_k\|_Q^2 + \frac{\eta}{2}\langle \lambda_k, e''(u_k,q_k)(u-u_k, q-q_k)^2\rangle \\ & + \langle \lambda, e'(u_k,q_k)(u-u_k,q-q_k) + e(u_k,q_k) - f\rangle,\end{aligned} \tag{5.23}$$

where $\eta = 1$ in the case of Method 2 (IRSQP method) and $\eta = 0$ for Method 3 (LMSQP method). The solutions $(\overline{u}_k, \overline{q}_k, \overline{\lambda}_k)$ of the quadratic programming problems are saddle points of the Lagrangian $\mathcal{L}_k$ (cf. ZEIDLER [160, p. 392ff]), i.e.,

$$\mathcal{L}_k(\overline{u}_k, \overline{q}_k, \lambda) \leq \mathcal{L}_k(\overline{u}_k, \overline{q}_k, \overline{\lambda}_k) \leq \mathcal{L}_k(u, q, \overline{\lambda}_k), \qquad \forall\ (u, q, \lambda) \in U \times Q \times Y^*, \tag{5.24}$$

and satisfy the optimality condition

$$0 = \mathcal{L}_k'(\overline{u}_k, \overline{q}_k, \overline{\lambda}_k), \tag{5.25}$$

where $\mathcal{L}_k'$ denotes the Fréchet-derivative of $\mathcal{L}_k$ in $U \times Q \times Y^*$.

In order to rewrite (5.25) as a linear system for (u, q, λ), the so-called *Karush-Kuhn-Tucker system*, we define the following operators:

$$\begin{aligned}
&K_k : U \to Y, && K_k u = e_u(u_k, q_k)u, \quad \forall\ u \in U, && (5.26)\\
&L_k : Q \to Y, && L_k q = e_q(u_k, q_k)q, \quad \forall\ q \in Q, && (5.27)\\
&M_k : U \to U^*, && \langle M_k u, v\rangle = \langle e_{uu}(u_k, q_k)(u, v), \lambda_k\rangle, \quad \forall\ (u, v) \in U \times U, && (5.28)\\
&N_k : Q \to Q^*, && \langle N_k q, s\rangle = \langle e_{qq}(u_k, q_k)(q, s), \lambda_k\rangle, \quad \forall\ (q, s) \in Q \times Q, && (5.29)\\
&P_k : U \to Q^*, && \langle P_k u, q\rangle = \langle e_{qu}(u_k, q_k)(q, u), \lambda_k\rangle, \quad \forall\ (u, q) \in U \times Q && (5.30)
\end{aligned}$$

Using these operators and the notation I_Q for the identity on Q, we may conclude that $(u_{k+1} - u_k, q_{k+1} - q_k, \lambda_{k+1})$ solves the linear system

$$\begin{pmatrix} E^*E + \eta M_k & \eta P_k^* & K_k^* \\ \eta P_k & \beta_k I_Q + \eta N_k & L_k^* \\ K_k & L_k & 0 \end{pmatrix} \begin{pmatrix} u \\ q \\ \lambda \end{pmatrix} = \begin{pmatrix} E^*(z^\delta - E u_k) \\ 0 \\ f - e(u_k, q_k) \end{pmatrix}. \tag{5.31}$$

Note that assumption (5.22) implies that K_k is a regular operator, while L_k is not necessarily invertible.

In the last part of this section we want to analyze the Karush-Kuhn-Tucker system (5.31). As it forms a symmetric, and indefinite linear system of equations, the analysis can be done in the framework of linear saddle-point problems.

The solution and numerical approximation of linear saddle-point problems arising from Lagrangian multipliers have been well-studied over the last decades after the seminal paper by BREZZI [30]. In the following let X and Λ be two Hilbert spaces, let $g \in X^*$, $f \in \Lambda^*$ and let $a : X \times X \to \mathbb{R}$ and $b : X \times \Lambda \to \mathbb{R}$ be continuous bilinear forms. Then a symmetric linear saddle-point problem in variational formulation consists of searching for a solution $(x, \lambda) \in X \times \Lambda$ of

$$a(x, v) + b(v, \lambda) = \langle g, v\rangle, \qquad \forall\ v \in X, \tag{5.32}$$

$$b(x, \mu) = \langle f, \mu\rangle, \qquad \forall\ \mu \in \Lambda, \tag{5.33}$$

where $a(\cdot,\cdot)$ is supposed to be symmetric on $X \times X$. The well-posedness of (5.32), (5.33) can be studied under additional assumptions on a and b, namely the so-called *kernel-ellipticity* of a,

$$\exists\ \alpha_a \in \mathbb{R}^+ :\ a(v, v) \geq \alpha_a \|v\|_X^2, \qquad \forall\ v \in K^b := \{v \in X \mid b(v, \mu) = 0, \forall\ \mu \in \Lambda\}, \tag{5.34}$$

and the *LBB-condition* upon b,

$$\exists\ \alpha_b \in \mathbb{R}^+ :\ \inf_{\mu\in\Lambda}\sup_{v\in X}\frac{b(v,\mu)}{\|v\|_X\|\mu\|_\Lambda} \geq \alpha_b. \tag{5.35}$$

Under these assumptions, the following classical result can be shown (cf. Brezzi [30] or Brezzi and Fortin [31]):

Theorem 5.2. *Let a, b as above, such that* (5.34) *and* (5.35) *are satisfied. Then the linear saddle-point problem* (5.32), (5.33) *has a unique solution $(x,\lambda) \in X \times \Lambda$, which depends continuously on the data $(f,g) \in \Lambda^* \times X^*$.*

In order to apply the above theorem we define the symmetric bilinear form a_k on $(U \times Q) \times (U \times Q)$ by

$$a_k(u,q;\varphi,\sigma) := \langle Eu, E\varphi\rangle_Z + \beta_k\langle q,\sigma\rangle_Q + \eta\big(\langle\varphi, M_k u\rangle + \langle\sigma, N_k q + P_k u\rangle + \langle q, P_k\varphi\rangle\big) \tag{5.36}$$

and the bilinear form $b_k : (U \times Q) \times Y^* \to \mathbb{R}$ by

$$b_k(u,q;\lambda) := \langle K_k u,\lambda\rangle + \langle L_k q,\lambda\rangle. \tag{5.37}$$

With the right-hand sides

$$\begin{aligned} f_k &= f - e(u_k,q_k) && \in Y,\\ g_k &= (E^*(z^\delta - Eu_k),0) && \in U^*\times Q^*, \end{aligned} \tag{5.38}$$

we can now rewrite the system (5.31) in the standard form

$$a_k(u,q;\varphi,\sigma) + b_k(\varphi,\sigma;\lambda) = \langle g_k,(\varphi,\sigma)\rangle, \qquad \forall\ (\varphi,\sigma)\in U\times Q, \tag{5.39}$$

$$b_k(u,q;\mu) = \langle f_k,\mu\rangle, \qquad \forall\ \mu \in Y^*. \tag{5.40}$$

Using the abstract theory of linear saddle point problems presented above, we can derive a statement on the well-posedness of the linear saddle-point problem (5.39), (5.40):

Theorem 5.3. *Suppose that $\eta = 0$ in* (5.36) *and the assumptions of Proposition 5.1 are satisfied. Then the indefinite system* (5.39), (5.40), *with the bilinear forms a_k and b_k defined via* (5.36), (5.37), *has a unique solution $(u,q,\lambda) \in U \times Q \times Y^*$, which depends continuously on the right-hand sides $f_k \in Y$ and $g_k \in U^* \times Q^*$.*

Proof. We first show the kernel-ellipticity (5.34) of a_k. Suppose (u,q) is an element of the null-space of b_k, then $u = -K_k^{-1}L_k q$ and thus, with $\eta = 0$, we may deduce that

$$\begin{aligned} a_k(u,q;u,q) &\geq \beta_k\|q\|_Q^2\\ &\geq \epsilon(\|u\|^2 + \|q\|^2) \end{aligned}$$

for some $\epsilon > 0$. The LBB-condition (5.35) for b_k follows from

$$\inf_{\lambda\in Y^*}\sup_{(u,q)\in U\times Q}\frac{b_k(u,q;\lambda)}{\|(u,q)\|\|\lambda\|} \geq \inf_{\lambda\in Y^*}\frac{b_k(K_k^{-1}\lambda,0;\lambda)}{\|K_k^{-1}\lambda\|\|\lambda\|} = \inf_{\lambda\in Y^*}\frac{\|\lambda\|^2}{\|K_k^{-1}\lambda\|\|\lambda\|} \geq \frac{1}{\|K_k^{-1}\|}.$$

Since the continuity of a_k and b_k follows from the continuity of the Fréchet-derivatives, Theorem 5.2 implies the assertion. ☐

The well-posedness of the linear saddle-point problem for $\eta = 1$ which corresponds to Method 2 can be shown in a similar way. But similar to the well-posedness result of the corresponding quadratic-programming problem it requires additional assumptions on the regularization parameter β_k. For details see Burger and Mühlhuber [33].

5.3.4 Comparison to the feasible path method

In the following we compare the behavior of the LMSQP-iteration with the feasible path approach presented in Chapter 4. This shall illustrate the differences in the iterates of the iteration in the product space compared to the ones generated using a solution operator for the state equation and optimizing only in the design space. The feasible path method looks as follows:

Method 4 (Feasible-Path Levenberg-Marquardt Method). *Let $q_0 \in Q$ be a given initial value and let $(\beta_k)_{k\in\mathbb{N}}$ be a bounded sequence of positive real numbers. The* Feasible-Path Levenberg-Marquardt *method consists of the iteration procedure*

$$q_{k+1} = \overline{q}_k, \tag{5.41}$$

where $\overline{q}_k \in Q$ is the minimizer of the quadratic programming problem

$$\frac{1}{2}\|ES(q_k) + ES_{lin}(q_k)(q - q_k) - z^\delta\|_Z^2 + \frac{\beta_k}{2}\|q - q_k\|_Q^2 \to \min_{q\in Q}, \tag{5.42}$$

with S denoting the solution operator of the state equation and S_{lin} denoting the solution operator of the linearized state equation.

For the sake of comparison we consider the LMSQP method in the parameter space, i.e., after elimination of the state u_{k+1} and the Lagrange parameter λ_{k+1}, which is possible because of the regularity of K_k (see (5.22)). For a better distinction, we denote the updates of the LMSQP-method by superscript SQP and those of the feasible path method by superscript FP.

The updates u^{SQP} and λ^{SQP} in the LMSQP-method can be computed consecutively from q^{SQP} via

$$\begin{aligned} u^{SQP} &= -K_k^{-1}\left(L_k q^{SQP} - f + e(u_k, q_k)\right) && (5.43)\\ \lambda^{SQP} &= -(K_k^*)^{-1}E^*\left(Eu^{SQP} - z^\delta + Eu_k\right) && (5.44)\end{aligned}$$

Thus, with the notation $G_k := -EK_k^{-1}L_k$, we may rewrite the optimality condition for the update q^{SQP} as

$$(\beta_k I_Q + G_k^* G_k)\; q^{SQP} = G_k^*(z^\delta - Eu_k - EK_k^{-1}(f - e(u_k, q_k)). \tag{5.45}$$

For a first comparison with the feasible path method we assume that $(u_0, q_0) \in U \times Q$ solves (5.7), i.e. the initial iterate is feasible with respect to the state equation. Then the iteration step q^{FP} for the feasible-path method solves

$$(\beta_0 I_Q + G_0^* G_0)\; q^{FP} = G_0^*(z^\delta - Eu_0), \tag{5.46}$$

which coincides with (5.45) in our particular case, i.e., the iterate q_1 computed with the LMSQP-method is the same as with the feasible path method. The difference of our SQP approach to the classical method following the feasible path occurs in the second step of the iteration, since u_1 is not on the feasible path anymore. If e' is Lipschitz-continuous, we only have

$$\|u_1^{SQP} - u_1^{FP}\|_U = \mathcal{O}(\|u_1^{FP} - u_0\|_U^2 + \|q_1^{SQP} - q_0\|_Q^2). \tag{5.47}$$

From equation (5.45) one observes that the right-hand side differs from the one for a feasible path approach, since $f - e(u_k, q_k)$ need not vanish, which is the essential difference of the LMSQP-method and the feasible path method.

Since the linear systems (5.45) and (5.46) are of the same structure, one could also think of finding a product-space formulation of the feasible path method. Such an approach was presented by Tautenhahn and Schweigert [151] also in the context of inverse problems, but in connection with a different regularization scheme (Tikhonov regularization). In order to derive such a formulation we introduce the state u_k which is the unique solution of $e(u, q_k) = f$ (for given q_k), and an auxiliary function w such that $ES_{\text{lin}}(q_k)(q - q_k) = E(w - u_k)$. Then from the definition of S_{lin} as a solution operator of the linearized state problem we may conclude that $e'(u_k, q_k)(w - u_k, q - q_k) = 0$. Since the pair (u_k, q_k) satisfies (5.7), we may add the term $e(u_k, q_k)$ and obtain that q_{k+1} is determined as the minimizer of

$$\frac{1}{2}\|Ew - z^\delta\|_Z^2 + \frac{\beta_k}{2}\|q - q_k\|_Q^2 \to \min_{(w,q)\in U\times Q}, \tag{5.48}$$

subject to the linear constraint

$$e(u_k, q_k) + e'(u_k, q_k)(w - u_k, q - q_k) = f. \tag{5.49}$$

The new iterate u_{k+1} can be computed subsequently as the solution of $e(u, q_{k+1}) = f$.

Since the minimization step yielding q_{k+1} is the same as a step in the LMSQP-method, we may interpret the feasible path method as a predictor-corrector variant of the LMSQP method, where the LMSQP-step serves as a predictor and a corrector step back to the feasible path is performed for fixed parameter q_{k+1}. In this formulation, the only difference of the LMSQP-method with respect to the feasible path scheme is to avoid the corrector step, which might be superfluous for many applications. Thus, the LMSQP-method may save some numerical advantage by avoiding the solution of possibly nonlinear state equations.

5.4 Discretization techniques

In the following we investigate the discretization of the LMSQP-method by a Galerkin approach. First of all, we assume that we have discretized data $z^{\delta,\eta} \in Z_\eta \subset Z$ of the form

$$z^{\delta,\eta} = R_\eta z^\delta, \tag{5.50}$$

where $R_\eta : Z \to Z_\eta$ is the orthogonal projector onto the finite-dimensional subspace Z_η. Note that we can give an error estimate for $z^{\delta,\eta}$ using (5.9) and $\|R_\eta\| = 1$, which yields

$$\delta_\eta := \|R_\eta z^\delta - z\|_Z \le \|R_\eta(z^\delta - z)\|_Z + \|R_\eta z - z\|_Z \le \delta + \inf_{y\in Z_\eta} \|y - z\|_Z. \tag{5.51}$$

Additionally we assume, that U is a Hilbert space and that the image space of e can be identified with the dual of U, for which reason we write U^* instead of Y in the following. Finally, we assume that e is continuously Frèchet-differentiable on $U \times Q$ and that the partial derivative e_u is self-adjoint and satisfies the coercivity condition

$$\langle e_u(u, q)v, v\rangle \ge \alpha_e \|v\|_U^2, \qquad \forall (u, q, v) \in U \times Q \times U, \tag{5.52}$$

for some $\alpha_e \in \mathbb{R}^+$.

The above setup is typical for a partial differential equation of elliptic type, which is also the main type of application we have in mind. We want to mention that the infinite-dimensional analysis carried out in the previous section was not restricted to elliptic problems, but only assumed well-posedness of the state equation for given parameter. However, since the numerical approximation techniques for elliptic problems differ from the ones for parabolic or hyperbolic problems (cf. e.g. Quarteroni and Valli [123] for an overview), one cannot expect a successful unified approach to corresponding parameter identification problems. For this reason we restrict ourselves to the investigation of the elliptic case.

5.4.1 The discretized LMSQP method and its well-posedness

Now let $U_h \subset U$, $Q_h \subset Q$ be finite-dimensional subspaces of U and Q, with the corresponding orthogonal projectors $P_h : U \to U_h$ and $\tilde{P}_h : Q \to Q_h$. Then we can discretize the LMSQP-Method as follows:

Method 5 (Galerkin LMSQP-Method). *Let U_h, Q_h and Z_η be as above and let $(u_0, q_0) \in U_h \times Q_h$ be a given initial value. Moreover, let $(\beta_k)_{k\in\mathbb{N}}$ be a bounded sequence of positive real numbers. The* Galerkin Levenberg-Marquardt sequential quadratic programming *(GLMSQP) method consists of the iteration procedure*

$$(u_{k+1}, q_{k+1}) = (\overline{u}_k, \overline{q}_k), \tag{5.53}$$

where $(\overline{u}_k, \overline{q}_k) \in U_h \times Q_h$ is the minimizer of the quadratic programming problem

$$\frac{1}{2}\|R_\eta(Eu - z^\delta)\|_Z^2 + \frac{\beta_k}{2}\|q - q_k\|_Q^2 \to \min_{(u,q)\in U_h\times Q_h}, \tag{5.54}$$

subject to the linear constraint

$$\langle e(u_k, q_k) + e'(u_k, q_k)(u - u_k, q - q_k), \varphi\rangle = \langle f, \varphi\rangle, \qquad \forall\, \varphi \in U_h. \tag{5.55}$$

Note that the constraint (5.55) can be rewritten in operator form as

$$P_h^* K_k P_h(u - u_k) + P_h^* L_k \tilde{P}_h(q - q_k) = P_h^*(f - e(u_k, q_k)), \tag{5.56}$$

to be solved for $(u, q) \in U_h \times Q_h$, with the notation

$$K_k : U \to U^*, \qquad K_k u = e_u(u_k, q_k)u, \quad \forall\, u \in U, \tag{5.57}$$

$$L_k : Q \to U^*, \qquad L_k q = e_q(u_k, q_k)q, \quad \forall\, q \in Q, \tag{5.58}$$

and $P_h^* : U_h^* \to U^*$ is the adjoint of P_h. Under the assumption (5.52), we obtain that

$$\langle P_h^* K_k P_h v, v\rangle = \langle K_k P_h v, P_h v\rangle = \langle K_k v, v\rangle \geq \alpha_e \|v\|_U^2 \tag{5.59}$$

for all $v \in U_h$, i.e., the discrete bilinear form associated with the operator $P_h^* K_k P_h$ is coercive on U_h. This implies by the Lax-Milgram theorem, that (5.56) is uniquely solvable with respect to u for given $q \in Q_h$. Consequently, in an analogous way to the proof of Proposition 5.1 we may show the following result on the well-posedness of the quadratic programming problem that has to be solved in each step of Method 5 (GLMSQP method).

Proposition 5.4. *Let e be continuously Frèchet-differentiable, let (5.52) hold and let $\beta_k > 0$. Then the quadratic programming problem (5.54), (5.55) has a unique solution $(\overline{u}_k, \overline{q}_k) \in U_h \times Q_h$, which is also the only local minimum.*

5.4.2 The discretized Karush-Kuhn-Tucker system

In Subsection 5.3.3, the Karush-Kuhn-Tucker system for the infinite-dimensional version of the LMSQP-method has been derived and analyzed in the framework of linear saddle point problems. Now we will discuss the discretized analogue of this system, namely the first-order optimality conditions for the quadratic programming problem (5.54), (5.55).

The Lagrangian of (5.54), (5.55) is given by

$$\begin{aligned}\mathcal{L}_k(u,q;\lambda) = &\frac{1}{2}\|R_\eta(Eu - z^\delta)\|_Z^2 + \frac{\beta_k}{2}\|q - q_k\|_Q^2 + \\ &+ \langle \lambda, e'(u_k,q_k)(u-u_k, q-q_k) + e(u_k,q_k) - f\rangle,\end{aligned} \tag{5.60}$$

for $(u,q,\lambda) \in U_h \times Q_h \times U_h$. Since P_h and $\tilde{P}_h$ are equal to the identity on U_h and Q_h, respectively, we can rewrite the Lagrangian as

$$\begin{aligned}\mathcal{L}_k(u,q;\lambda) = &\frac{1}{2}\|R_\eta(EP_hu - z^\delta)\|_Z^2 + \frac{\beta_k}{2}\|\tilde{P}_h(q - q_k)\|_Q^2 + \\ &+ \langle P_h\lambda, K_kP_h(u-u_k) + L_k\tilde{P}_h(q-q_k) + e(u_k,q_k) - f\rangle,\end{aligned} \tag{5.61}$$

with the operators K_k and L_k defined by (5.57), (5.58). The KKT-system can now be deduced by computing the partial derivatives of the Lagrangian with respect to u, q and λ, i.e., $(u_{k+1} - u_k, q_{k+1} - q_k, \lambda_{k+1})$ solves the linear saddle-point problem

$$\begin{pmatrix} P_h^*E^*R_\eta^*R_\eta EP_h & 0 & P_h^*K_k^*P_h \\ 0 & \beta_k\tilde{P}_h^*\tilde{P}_h & \tilde{P}_h^*L_k^*P_h \\ P_h^*K_kP_h & P_h^*L_k\tilde{P}_h & 0 \end{pmatrix} \begin{pmatrix} u \\ q \\ \lambda \end{pmatrix} = \begin{pmatrix} P_h^*E^*R_\eta^*R_\eta(z^\delta - Eu_k) \\ 0 \\ P_h^*(f - e(u_k,q_k)) \end{pmatrix}. \tag{5.62}$$

As in Subsection 5.3.2, we define the symmetric bilinear form $a_k : (U \times Q) \times (U \times Q) \to \mathbb{R}$ by

$$a_k^\eta(u,q;\varphi,\sigma) := \langle R_\eta Eu, R_\eta E\varphi\rangle_Z + \beta_k\langle q,\sigma\rangle_Q \tag{5.63}$$

and the bilinear form $b_k : (U \times Q) \times U \to \mathbb{R}$ by

$$b_k(u,q;\lambda) := \langle K_ku,\lambda\rangle + \langle L_ku,\lambda\rangle. \tag{5.64}$$

Moreover, we use the right-hand sides

$$f_k \;:=\; f - e(u_k,q_k) \quad \in U^*, \tag{5.65}$$

$$g_k^\eta \;:=\; (E^*R_\eta^*R_\eta(z^\delta - Eu_k), 0) \quad \in U^* \times Q. \tag{5.66}$$

Then the KKT-system (5.62) can be interpreted as the Galerkin approximation of an indefinite variational problem, i.e., $(u,q,\lambda) \in U_h \times Q_h \times U_h$ is the solution of

$$a_k^\eta(u,q;\varphi,\sigma) + b_k(\varphi,\sigma;\lambda) = \langle g_k^\eta, (\varphi,\sigma)\rangle, \qquad \forall\ (\varphi,\sigma) \in U_h \times Q_h, \tag{5.67}$$

$$b_k(u,q;\mu) = \langle f_k,\mu\rangle, \qquad \forall\ \mu \in U_h. \tag{5.68}$$

In an analogous way to the proof of Theorem 5.2 we can show that the bilinear form a_k^η satisfies the discrete kernel-ellipticity condition on $U_h \times Q_h$, i.e., there exists a constant $\alpha_a > 0$ such that

$$a_k^\eta(u,q;u,q) \geq \alpha_a\|(u,q)\|^2, \quad \forall\ (u,q) \in \mathcal{K}_b^h$$

with

$$\mathcal{K}_b^h := \{(v,s) \in U_h \times Q_h \mid b(v,s;\lambda) = 0,\ \forall\ \lambda \in U_h\},$$

and that b satisfies the discrete LBB-condition

$$\inf_{\lambda \in U_h} \sup_{(u,q)\in U_h \times Q_h} \frac{b_k(u,q;\lambda)}{\|(u,q)\|\ \|\lambda\|} \geq \alpha_b,$$

for some $\alpha_b > 0$. Using the Theorem 5.2 this implies the following well-posedness result for the discretized problem (5.67), (5.68):

Theorem 5.5. *Let e be continuously Frèchet-differentiable, let (5.52) hold and let $\beta_k > 0$. Then the indefinite system (5.67), (5.68) has a unique solution $(u,q,\lambda) \in U_h \times Q_h \times U_h$, which depends continuously on the right-hand sides f_k and g_k^η.*

Since the constants α_a and α_b are the same as in the corresponding infinite-dimensional conditions in $U \times Q$, they are in particular independent of the discrete subspaces U_h and Q_h. This allows us to deduce an approximation result for the solutions of (5.67), (5.68) to the solution $(u,q,\lambda) \in U \times Q \times U$ of the infinite-dimensional KKT-System, given in variational form in (5.39), (5.40), i.e.

$$\begin{aligned} a_k(u,q;\varphi,\sigma) + b_k(\varphi,\sigma;\lambda) &= \langle g_k, (\varphi,\sigma)\rangle, && \forall\ (\varphi,\sigma) \in U \times Q, && (5.69)\\ b_k(u,q;\mu) &= \langle f_k, \mu\rangle, && \forall\ \mu \in U, && (5.70)\end{aligned}$$

with a_k given by

$$a_k(u,q;\varphi,\sigma) := \langle Eu, E\varphi\rangle_Z + \beta_k \langle q,\sigma\rangle_Q, \tag{5.71}$$

b_k, f_k as above and g_k defined by

$$g_k := (E^*(z^\delta - Eu_k), 0) \in U^* \times Q. \tag{5.72}$$

Theorem 5.6. *Suppose that the assumptions of Theorem 5.5 are satisfied and let*

$$(u_h, q_h, \lambda_h) \in U_h \times Q_h \times U_h$$

denote the unique solution of (5.67), (5.68). Then there exists a constant $c > 0$ independent of U_h and Q_h such that

$$\|(u-u_h, q-q_h, \lambda-\lambda_h)\| \leq c\left(r_{\eta,h}^\delta + \inf_{(v,s,\mu)\in U_h\times Q_h \times U_h} \|(u-v, q-s, \lambda-\mu)\|\right), \tag{5.73}$$

where (u,q,λ) denotes the unique solution of (5.39), (5.40) and

$$r_{\eta,h}^\delta := \|(R_\eta - I)z^\delta\|_Z + \sup_{v\in U_h, \|v\|=1} \|(R_\eta - I)Ev\|_Z. \tag{5.74}$$

Proof. First, let $(\tilde{u}_h, \tilde{q}_h, \tilde{\lambda}_h)$ denote the solution of (5.67), (5.68) with a_k^η, g_k^η replaced by a_k, g_k. Then Theorem 2.1 in Brezzi and Fortin [31] implies the existence of a constant $c_1 > 0$ (independent of U_h and Q_h) such that

$$\|(u-\tilde{u}_h, q-\tilde{q}_h, \lambda-\tilde{\lambda}_h)\| \leq c_1 \inf_{(v,s,\mu)\in U_h\times Q_h\times U_h} \|(u-v, q-s, \lambda-\mu)\|.$$

Moreover, the continuous dependence of the solutions of (5.67), (5.68) on the right-hand side implies the existence of $c_2 > 0$ with

$$\begin{aligned}
&\|(u_h - \tilde{u}_h, q_h - \tilde{q}_h, \lambda_h - \tilde{\lambda}_h)\| \\
&\quad \le c_2 \left(\sup_{v \in U_h, \|v\|=1} \langle g_k^\eta - g_k, (v,0) \rangle + \sup_{\varphi \in U_h, \|\varphi\|=1} |a_k^\eta(\tilde{u}_h, \tilde{q}_h, \varphi) - a_k(\tilde{u}_h, \tilde{q}_h, \varphi)| \right) \\
&\quad \le c_2 \left(\sup_{v \in U_h, \|v\|=1} \langle Ev, (R_\eta^* R_\eta - I)(z^\delta - Eu_k) \rangle + \sup_{\varphi \in U_h, \|\varphi\|=1} \langle E\varphi, (R_\eta^* R_\eta - I) E\tilde{u}_h \rangle \right) \\
&\quad \le c_2 \, \|E\| \, \|(R_\eta - I) z^\delta\|_Z + c_3 \sup_{v \in U_h, \|v\|=1} \|(R_\eta - I) Ev\|_Z,
\end{aligned}$$

and with the triangle inequality we may conclude (5.73). □

Theorem 5.6 provides an error estimate for the solutions of the discretized saddle-point problem (5.67), (5.68), consisting of two parts corresponding to the numerical approximation in the image space Z and in the pre-image spaces U and Q. An obvious estimate for the first term is

$$r_{\eta,h}^\delta \le \inf_{y \in Z_\eta} \|y - z^\delta\|_Z + \sup_{v \in U_h, \|v\|_U = 1} \inf_{\tilde{y} \in Z_\eta} \|\tilde{y} - Ev\|_Z,$$

which possibly does not lead to a quantitative estimate, since there is no additional information on the smoothness of the noisy data. An alternative estimate is

$$r_{\eta,h}^\delta \le \delta + \inf_{y \in Z_\eta} \|y - z\|_Z + \sup_{v \in U_h, \|v\|_U = 1} \inf_{\tilde{y} \in Z_\eta} \|\tilde{y} - Ev\|_Z.$$

The infimum of $\|y - z\|_Z$ can usually be estimated more easily, since the exact data z are smoother due to the fact that $\hat{u}$ is the solution of the state equation for some parameter $\hat{q}$. E.g., if the state equation is of elliptic type with solution $\hat{u} \in H^1(\Omega)$, $E : H^1(\Omega) \to L^2(\Omega)$ is the embedding operator, and R_η results from a standard finite element discretization on a grid with fineness η, then we have at least

$$\inf_{y \in Z_\eta} \|y - z\| = \mathcal{O}(\eta).$$

Another important observation is that the last term vanishes if the discrete spaces Z_η and U_h are equal, which can be achieved in some applications.

The second term in (5.73) shows that the Galerkin approximation of the KKT-system is of optimal order in $U_h \times Q_h \times U_h$; it can be estimated by standard methods for finite element discretizations; quantitative estimates can be obtained using the regularity of the iterates. This part depends of course strongly on the specific application.

5.5 Numerical realization of the SQP-iteration

In the following we want to discuss some numerical methods and variants for the 'outer iteration', i.e., the Galerkin LMSQP algorithm under the assumption that we are able to solve the discretized KKT-system numerically. The 'inner iteration', namely the numerical solution of the indefinite system (5.62) will be investigated in Section 5.6.

5.5.1 Scaling of state variable, parameter and Lagrangian multiplier

The performance of an iteration algorithm often depends crucially on the way the problem is formulated. Scaling is a well-known technique for reformulating an optimization problem whose main objective is twofold: On the one hand all the variables should be of similar magnitude, on the other hand also the value of the derivatives should all be of similar size. In unconstrained optimization, a problem should be rescaled in such a way, that changes of the iterate in one direction do not result in by far larger changes of the value of the objective than changes in another direction. In constrained optimization the above statements are also true for each constraint. Additionally the set of constraints should be well balanced with respect to each other such that each constraint has equal weight. Furthermore, the set of constraints should be balanced with respect to the objective. As scaling is of high practical importance for any optimization problem, many aspects can be found in monographs on optimization (cf. e.g. Gill, Murray, and Wright [63] or Nocedal and Wright [115]).

We want to consider only the last aspect in this context, i.e., the scaling of the state constraint with respect to the objective which is also of high importance for achieving fast convergence of the outer iteration. For the inner iteration, the aspect of scaling can be included in the construction of a good preconditioner. The outer iteration of an SQP method tries to attain two goals at the same time: Feasibility of the iterate with respect to the state constraint and optimality of the iterate with respect to the objective. One aspect dominating the other results usually in bad convergence properties: If the feasibility aspect dominates, only very small changes of the iterate are possible in order to ensure 'almost' feasibility. If the optimality aspect dominates, any violation of the state constraint is reduced too slowly.

For the LMSQP method in the form of (5.54), (5.55) it turned out that in many situations the feasibility aspect is strongly dominating. Using line search methods for globalization (see also Section 2.2) this results usually in step lengths much smaller than one. Replacing the state constraint by a preconditioned state constraint leads to a better balanced formulation and to much faster convergence. Furthermore a step length parameter equal to one is accepted in almost all steps.

5.5.2 Globalization strategies

The LMSQP method is a variant of Newton's method and therefore only locally convergent. For this reason, globalization strategies, such as *trust region methods* or *line search strategies* (which are the two most popular classes of globalization techniques in optimization), are needed. Both classes were introduced in Section 2.2. That is why we want to refer to this section and the references cited therein for details.

5.5.3 Nested multi-level optimization techniques

Important tools for the efficient numerical approximation of infinite-dimensional optimization problems are *multi-level optimization methods*. In the nested multi-level setup, one starts the optimization procedure at a coarse level $U_{h_1} \times Q_{h_1}$, where the iteration procedure can be carried out efficiently. If an appropriate stopping criterion is satisfied, one interpolates the state and parameter obtained in this way to a finer level $U_{h_2} \times Q_{h_2}$ (for $h_2 < h_1$), serving now as a starting value on this level. This procedure is repeated until the finest level is reached. Usually, nested spaces are used in this approach, i.e., $U_{h_1} \subset U_{h_2}$, $Q_{h_1} \subset Q_{h_2}$ (for $h_2 < h_1$), which leads to simple interpolation operators. Since one cannot choose the discretization of the data arbitrarily in general , we consider only the case of fixed η here, but a multi-level approach in η can be realized in an analogous way, if necessary.

Nested multi-level methods outperform standard discretization techniques in many cases (cf. e.g. Heinkenschloss [82], Heise [84], Lukáš [108, 109]); usually a considerable number of iterations is needed on the coarse level only, where the numerical effort per iteration is very low. On the finest levels, the stopping criterion is often satisfied already after one iteration step and so the overall effort is less than for a direct discretization on the finest level. For the Galerkin LMSQP method, this leads to Algorithm 5.5.3.

Algorithm 5.1 Nested Multi-Level Galerkin LMSQP

Require: a decreasing sequence $\{h_\ell\}_{\ell=1,\dots,L}$ with nested spaces $U_{h_\ell} \subset U_{h_{\ell+1}}$, $Q_{h_\ell} \subset Q_{h_{\ell+1}}$ (e.g. $h_\ell = 2^{-\ell} h_0$)
Require: $(u_0^1, q_0^1) \in U_{h_1} \times Q_{h_1}$
  for $\ell = 1$ to L **do**
    $h = h_\ell$
    Perform the Galerkin LMSQP method until the stopping criterion is satisfied.
    if $\ell == L$ **then**
      return
    end if
    Prolongate the iteration $(u_{k_*}^\ell, q_{k_*}^\ell)$ to the finer level $U_{h_{\ell+1}} \times Q_{h_{\ell+1}}$, which results in a new starting value $(u_0^{\ell+1}, q_0^{\ell+1})$.
  end for

Up to now we did not talk about the choice of the nested spaces. Of course, they can be chosen in advance. In the finite element community it is well known, that the accuracy of the solution can be improved by using a-posteriori error estimators. They provide information which elements shall be refined to obtain a more precise solution. This information can be used to construct appropriate fine grid spaces. For an overview of a-posteriori error estimation see e.g. Verfürth [154]. In the context of optimization the concept of adaptivity and a-posteriori error estimation is not as well-known as in the finite element community. An example in the context of optimization is presented in Becker, Kapp, and Rannacher [11], in the context of optimal control problems see e.g. Becker, Kapp, and Rannacher [10].

5.6 Numerical solution of the KKT-system

In the following we will discuss the numerical solution of the discretized KKT-system (5.62) for fixed iteration number k. We have seen above that the Galerkin-type approximation (5.62) of the original KKT-system is well-posed, now we discuss some of its structural properties, which are important for the application of iterative solution methods and for the construction of preconditioners.

Choosing bases

$$\Phi = (\phi_1, \dots, \phi_m)^T \in U_h, \qquad \Sigma = (\sigma_1, \dots, \sigma_n)^T \in Q_h, \tag{5.75}$$

of the finite-dimensional subspaces U_h and Q_h, we may represent $(u_h, q_h, \lambda_h) \in U_h \times Q_h \times U_h$ via

$$u_h = \mathbf{u}^T \Phi, \qquad q_h = \mathbf{q}^T \Sigma, \qquad \lambda_h = \boldsymbol{\lambda}^T \Phi, \tag{5.76}$$

with coordinate vectors $\mathbf{u}, \boldsymbol{\lambda} \in \mathbb{R}^m$ and $\mathbf{q} \in \mathbb{R}^n$. In order to transform (5.62) into a linear system for $\mathbf{u}$, $\mathbf{q}$ and $\boldsymbol{\lambda}$, we define the matrices

$$\mathbf{G} := (\langle E\phi_j, E\phi_i\rangle_Z)_{i,j=1,\dots,m} \qquad \mathbf{H} := (\langle \sigma_j, \sigma_i\rangle_Q)_{i,j=1,\dots,n} \tag{5.77}$$

$$\mathbf{K} := (\langle K_k\phi_j, \phi_i\rangle)_{i,j=1,\dots,m} \qquad \mathbf{L} := (\langle L_k\sigma_j, \phi_i\rangle)_{i=1,\dots,m;j=1,\dots,n} \tag{5.78}$$

and the vectors

$$\mathbf{f}_1 := (\langle z^{\delta,\eta} - Eu_k, E\varphi_i\rangle_Z)_{i=1,\dots,m}, \qquad \mathbf{f}_3 := (\langle f - e(u_k, q_k), \varphi_i\rangle)_{i=1,\dots,m}. \tag{5.79}$$

This allows us to rewrite the discretized KKT-system (with penalty parameter $\beta = \beta_k$) as

$$\begin{pmatrix} \mathbf{G} & 0 & \mathbf{K}^T \\ 0 & \beta\mathbf{H} & \mathbf{L}^T \\ \mathbf{K} & \mathbf{L} & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{q} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ 0 \\ \mathbf{f}_3 \end{pmatrix}, \tag{5.80}$$

respectively as

$$\mathbf{M}\,\mathbf{X} = \mathbf{F}, \tag{5.81}$$

with

$$\mathbf{M} = \begin{pmatrix} \mathbf{G} & 0 & \mathbf{K}^T \\ 0 & \beta\mathbf{H} & \mathbf{L}^T \\ \mathbf{K} & \mathbf{L} & 0 \end{pmatrix}, \qquad \mathbf{X} = \begin{pmatrix} \mathbf{u} \\ \mathbf{q} \\ \boldsymbol{\lambda} \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} \mathbf{f}_1 \\ 0 \\ \mathbf{f}_3 \end{pmatrix}.$$

The structural properties of $\mathbf{M}$ and its sub-matrices will be examined in the following section.

5.6.1 The system matrix M

Due to the well-posedness result on the discretized KKT-system (5.62) (cf. Theorem 5.5), we may conclude that the system matrix $\mathbf{M}$ is regular. In order to obtain further insight into the structure of $\mathbf{M}$, we investigate the properties of the sub-matrices $\mathbf{G}$, $\mathbf{H}$, $\mathbf{K}$ and $\mathbf{L}$.

Proposition 5.7. *The matrices $\mathbf{K} \in \mathbb{R}^{m\times m}$ and $\mathbf{H} \in \mathbb{R}^{n\times n}$ are symmetric positive definite, and the matrix $\mathbf{G} \in \mathbb{R}^{m\times m}$ is symmetric positive semi-definite. If in addition the operator E is injective on U_h, then $\mathbf{G}$ is regular, too.*

Proof. Let u_h and q_h be as in (5.76), then there exist constants $c_1(h)$ and $c_2(h)$ such that

$$\|u_h\|_U \geq c_1(h)\|\mathbf{u}\|, \qquad \|q_h\|_Q \geq c_2(h)\|\mathbf{q}\|,$$

where $\|.\|$ denotes the Euclidian norm in $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively. Thus, we have

$$\mathbf{u}^T\mathbf{K}\,\mathbf{u} = \langle K_k u_h, u_h\rangle \geq \alpha_e\|u_h\|_U^2 \geq \alpha_e c_1(h)^2\|\mathbf{u}\|^2,$$

and

$$\mathbf{q}^T\mathbf{H}\,\mathbf{q} = \|q_h\|_Q^2 \geq c_2(h)^2\|\mathbf{q}\|^2.$$

Moreover, the identity

$$\mathbf{u}^T\mathbf{G}\,\mathbf{u} = \|Eu_h\|_Z^2 \geq 0$$

implies that $\mathbf{G}$ is positive semi-definite and regular under the assumption that E is injective on U_h. The symmetry of the matrices $\mathbf{G}$, $\mathbf{H}$ and $\mathbf{K}$ can be verified in a similar way, using the symmetry of scalar products and the self-adjointness of the operator K_k. □

The matrix $\mathbf{L} \in \mathbb{R}^{m\times n}$ is difficult to analyze, it is neither symmetric nor regular in general (in particular if $n \neq m$). However, some fundamental properties of $\mathbf{M}$ (such as its regularity) rely rather on $\mathbf{G}$, $\mathbf{H}$ and $\mathbf{K}$ than on $\mathbf{L}$. Moreover, the classical splitting of a symmetric saddle-point problem as

$$\begin{pmatrix} \mathbf{G} & 0 & \mathbf{K}^T \\ 0 & \mathbf{H}_\beta & \mathbf{L}^T \\ \mathbf{K} & \mathbf{L} & 0 \end{pmatrix} = \begin{pmatrix} \mathbf{I} & 0 & 0 \\ 0 & \mathbf{I} & 0 \\ \mathbf{K}\mathbf{G}^{-1} & \mathbf{L}\mathbf{H}_\beta^{-1} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{G} & 0 & 0 \\ 0 & \mathbf{H}_\beta & 0 \\ 0 & 0 & -\mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{I} & 0 & \mathbf{G}^{-1}\mathbf{K}^T \\ 0 & \mathbf{I} & \mathbf{H}_\beta^{-1}\mathbf{L}^T \\ 0 & 0 & \mathbf{I} \end{pmatrix},$$

where $\mathbf{H}_\beta := \beta\mathbf{H}$ and $\mathbf{C}$ is the Schur-complement

$$\mathbf{C} := \mathbf{K}\mathbf{G}^{-1}\mathbf{K}^T + \beta^{-1}\mathbf{L}\mathbf{H}_\beta^{-1}\mathbf{L}^T, \tag{5.82}$$

is only possible if both $\mathbf{G}$ and $\mathbf{H}_\beta$ are regular. In particular, we may conclude that $\mathbf{M}$ has $n+m$ positive and m negative eigenvalues.

5.6.2 Reduced SQP approaches

The basic idea of reduced SQP-methods is the a-priori elimination of the equality constraint, which can be written in matrix form as

$$\mathbf{K}\mathbf{u} + \mathbf{L}\mathbf{q} = \mathbf{f}_3, \tag{5.83}$$

which is equivalent to an elimination of $\mathbf{u}$ and $\boldsymbol{\lambda}$ in (5.80).

Due to Proposition 5.7, $\mathbf{K}$ is a regular, symmetric matrix and thus, we may compute

$$\mathbf{u} = \mathbf{K}^{-1}(\mathbf{f}_3 - \mathbf{L}\mathbf{q}), \tag{5.84}$$
$$\boldsymbol{\lambda} = \mathbf{K}^{-T}(\mathbf{f}_1 - \mathbf{G}\mathbf{u}), \tag{5.85}$$

which yields after some calculations the $n \times n$-system

$$\mathbf{M}_r\mathbf{q} = \mathbf{g} \tag{5.86}$$

with

$$\mathbf{M}_r := \mathbf{H}_\beta + \mathbf{L}^T\mathbf{K}^{-T}\mathbf{G}\mathbf{K}^{-1}\mathbf{L} \tag{5.87}$$
$$\mathbf{g} := \mathbf{L}^T\mathbf{K}^{-T}(\mathbf{G}\mathbf{K}^{-1}\mathbf{f}_3 - \mathbf{f}_1). \tag{5.88}$$

The reduced SQP-approach seems of particular interest if $n \ll m$, which is a frequently used discretization strategy for parameter identification and optimal control problems (cf. e.g. Sachs [134], Schulz and Bock [139], or Schulz [138]). The original matrix $\mathbf{M}$ is an indefinite matrix of size $(2m+n)\times(2m+n)$, while the reduced system matrix $\mathbf{M}_r$ in (5.86) is of size $n\times n$. However, $\mathbf{M}_r$ is not a sparse matrix even if all the sub-matrices of $\mathbf{M}$ are sparse, since it involves the inverse of $\mathbf{K}$. Moreover, the evaluation of a matrix-vector product using $\mathbf{M}_r$ is more expensive than a matrix-vector product using $\mathbf{M}$, since it involves the solution of two systems of the form

$$\mathbf{K}\mathbf{w} = \mathbf{g}, \tag{5.89}$$

with different right-hand sides $\mathbf{g}$, while for the evaluation of matrix-vector product with $\mathbf{M}$ only direct evaluations of $\mathbf{K}$ are needed, which are very cheap for typical finite element discretization of the state constraint. In practice, one usually tries to compensate this disadvantage of reduced SQP-methods by using a Broyden-type update for the reduced system matrix instead of the exact matrix $\mathbf{M}_r$, which leads to efficient optimization algorithms for small numbers of design parameters n.

5.6.3 Simultaneous solution of the KKT-system

Recently, the simultaneous solution of KKT-systems by iterative methods has been investigated, in particular in connection with optimal control problems (cf. BATTERMANN AND HEINKENSCHLOSS [8], BIROS AND GHATTAS [16, 17] or HABER AND ASCHER [74]). Compared to the reduced SQP-approach, a simultaneous solution strategy has the obvious advantage that the allocation and evaluation of the system matrix $\mathbf{M}$ is much cheaper than of $\mathbf{M}_r$. The pay-off is that $\mathbf{M}$ is indefinite and larger than $\mathbf{M}_r$, which might cause additional effort. However, the main effort in the reduced SQP-approach is related to the evaluation or assembly of the system matrix $\mathbf{M}_r$, respectively, and therefore a simultaneous solution of the KKT-system can result in a tremendous speed-up of the SQP-method, in particular for fine discretizations.

At a first glance, it seems rather straight-forward to solve (5.81) by a standard iterative method for indefinite systems such as inexact Uzawa methods (cf. BRAMBLE, PASCIAK, AND VASSILEV [27], ELMAN AND GOLUB [49], LANGER AND QUECK [104, 105], or QUECK [124]) or Krylov-subspace methods such as GMRES (cf. SAAD AND SCHULTZ [133]), MINRES (cf. PAIGE AND SAUNDERS [117]) and QMR (cf. FREUND AND NACHTIGAL [56]). However, in the case of large-scale problems, we have to expect a large condition number (note that β is usually small and that $\mathbf{M}$ is singular for $\beta = 0$) and a complicated eigenvalue pattern of the matrix $\mathbf{M}$, which might cause iterative methods to diverge or to need a high number of iterations. Therefore, an appropriate preconditioning technique seems necessary for any of the methods.

In the following we distinguish two types of solvers that seem appropriate for the solution of the indefinite system (5.81) and discuss their basic properties with respect to the special structure of $\mathbf{M}$.

Inexact Uzawa iterations

Inexact Uzawa methods and similar iteration procedures have been developed for the solution of the classical Stokes system and similar problems (cf. QUARTERONI AND VALLI [123] for an overview). The classical Uzawa method is just a gradient method for the dual of the corresponding Lagrange functional, the inexact Uzawa method can be interpreted as a preconditioned version (cf. QUARTERONI AND VALLI [123]). Following the exposition by ZULEHNER [164], we can write an inexact Uzawa method for a system of the form (5.80) as

$$\hat{\mathbf{A}} \begin{pmatrix} \mathbf{u}_{k+1} - \mathbf{u}_k \\ \mathbf{q}_{k+1} - \mathbf{q}_k \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 - \mathbf{G}\mathbf{u}_k - \mathbf{K}\boldsymbol{\lambda}_k \\ -\beta\mathbf{H}\mathbf{q}_k - \mathbf{L}\boldsymbol{\lambda}_k \end{pmatrix}, \tag{5.90}$$

followed by

$$\hat{\mathbf{C}}(\boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_k) = \mathbf{f}_3 - \mathbf{K}\mathbf{u}_{k+1} - \mathbf{L}\mathbf{q}_{k+1}, \tag{5.91}$$

where $\hat{\mathbf{A}}$ is a preconditioner for the diagonal matrix

$$\mathbf{A} := \begin{pmatrix} \mathbf{G} & 0 \\ 0 & \beta\mathbf{H} \end{pmatrix}, \tag{5.92}$$

$\hat{\mathbf{C}}$ is a preconditioner for the Schur-complement $\mathbf{C}$ defined by (5.82) and k denotes the iteration index. In terms of (5.81) we can write the inexact Uzawa iteration as

$$\mathbf{X}_{k+1} = (\mathbf{I} - \hat{\mathbf{M}}^{-1}\mathbf{M})\mathbf{X}_k + \hat{\mathbf{M}}^{-1}\mathbf{F}, s \tag{5.93}$$

where $\hat{\mathbf{M}}$ is a preconditioner for the system matrix, given by

$$\hat{\mathbf{M}} = \begin{pmatrix} \hat{\mathbf{A}} & 0 \\ \mathbf{B} & \hat{\mathbf{C}} \end{pmatrix}, \tag{5.94}$$

with $\mathbf{B} = \begin{pmatrix} \mathbf{K} & \mathbf{L} \end{pmatrix}$.

A convergence analysis of this method is available only in the case when $\mathbf{A}$ is a regular matrix (cf. Bramble, Pasciak, and Vassilev [27] or Zulehner [164]), which means that we have to assume that $\mathbf{G}$ is regular. The latter is true e.g. if the data z represent distributed data for the state, i.e., E is an embedding operator. In this case, the structure of $\mathbf{A}$ is rather simple and it is not a difficult task to construct a preconditioner, even exact preconditioning seems possible (note that $\mathbf{G}$ is just a mass matrix for a typical finite element discretization). Since the matrices $\mathbf{G}$ and $\mathbf{H}$ do not change during the SQP-iteration we may even compute decompositions in a preprocessing step. The construction of a preconditioner for the Schur-complement $\mathbf{C}$ is more difficult and must take into account the specific nature of the underlying state equation.

Krylov-subspace methods

The Krylov-subspace methods GMRES and QMR are variants of the CG-algorithm that are applicable to indefinite problems, too. The basic idea of such methods is a defect minimization in the Krylov-subspace

$$\mathcal{K}_k(\mathbf{M}; \mathbf{X}_1) = \quad \{\mathbf{X}_1, \mathbf{M}\mathbf{X}_1, \dots, \mathbf{M}^{k-1}\mathbf{X}_1\}, \tag{5.95}$$

generated by $\mathbf{X}_1$, in the k-th iteration step. Since preconditioned CG-methods are probably the most successful class of iteration methods for positive definite systems, such methods seem very attractive also in the indefinite case, although additional difficulties may arise (cf. e.g. Saad and Schultz [133]).

The convergence analysis in Saad and Schultz [133] and Freund and Nachtigal [56] shows that the error bounds obtained for both methods are essentially the same, and mainly dependent on the eigenvalue distribution and the condition number of the system matrix $\mathbf{M}$. Therefore, appropriate preconditioning is again of high importance, in this case also with the possibility that $\mathbf{G}$ is singular.

Preconditioning

For the efficient solution of the KKT-system (5.80) it is necessary to use iterative solution procedures due to the size of the equation system. For these methods appropriate preconditioning strategies are needed to get fast convergence. Unfortunately, for symmetric indefinite equation systems by far fewer methods compared to the positive definite case are available.

The most popular class of methods are Uzawa type methods. Many publications can be found, especially in the field of fluid dynamics. Recently, Zulehner [164] presented a unified approach to many of these methods. The methodology presented in the previous subsection on inexact Uzawa methods can also be used as preconditioner.

A different class of preconditioners originates form reduced SQP methods and can be explained as follows: The KKT-matrix $\mathbf{M}$ can be factorized into

$$\mathbf{M} = \begin{pmatrix} \mathbf{G}\mathbf{K}^{-1} & 0 & \mathbf{I} \\ 0 & \mathbf{I} & \mathbf{L}^T\mathbf{K}^{-T} \\ \mathbf{I} & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{K} & \mathbf{L} & 0 \\ 0 & \mathbf{S_c} & 0 \\ 0 & -\mathbf{G}\mathbf{K}^{-1}\mathbf{L} & \mathbf{K}^T \end{pmatrix} \tag{5.96}$$

where $\mathbf{S_c}$ denotes the Schur-complement

$$\mathbf{S_c} = \beta\mathbf{H} + \mathbf{L}^T\mathbf{K}^{-T}\mathbf{G}\mathbf{K}^{-1}\mathbf{L}. \tag{5.97}$$

Replacing the matrix $\mathbf{K}^{-1}$ by a preconditioner $\hat{\mathbf{K}}^{-1}$ (e.g. a multigrid preconditioner) and the Schur complement $\mathbf{S_c}$ by an appropriate preconditioner $\hat{\mathbf{S}}_\mathbf{c}$ leads to a preconditioner for $\mathbf{M}$ of the form

$$\hat{\mathbf{M}} = \begin{pmatrix} \mathbf{G}\hat{\mathbf{K}}^{-1} & 0 & \mathbf{I} \\ 0 & \mathbf{I} & \mathbf{L}^T\hat{\mathbf{K}}^{-T} \\ \mathbf{I} & 0 & 0 \end{pmatrix} \begin{pmatrix} \hat{\mathbf{K}} & \mathbf{L} & 0 \\ 0 & \hat{\mathbf{S}}_\mathbf{c} & 0 \\ 0 & -\mathbf{G}\hat{\mathbf{K}}^{-1}\mathbf{L} & \hat{\mathbf{K}}^T \end{pmatrix}. \tag{5.98}$$

It must be noted that for the preconditioning operation $\hat{\mathbf{M}}^{-1}$ only applications of $\hat{\mathbf{K}}^{-1}$ and $\hat{\mathbf{S}}_\mathbf{c}^{-1}$ are necessary and no applications of $\hat{\mathbf{K}}$ of $\hat{\mathbf{S}}_\mathbf{c}$. This preconditioner was used in our computations (see Section 5.7) , but also by HABER AND ASCHER [74] or BIROS AND GHATTAS [16, 17].

A similar preconditioner was presented by BATTERMANN AND SACHS [9]. They used

$$\hat{\mathbf{M}} = \begin{pmatrix} 0 & 0 & \hat{\mathbf{K}}^T \\ 0 & \hat{\mathbf{S}}_\mathbf{c} & \mathbf{L}^T \\ \hat{\mathbf{K}} & \mathbf{L} & 0 \end{pmatrix} \tag{5.99}$$

as a preconditioner for an all-at-once approach for an optimal control problem. Their paper contains also some analysis of the eigenvalue structure of the preconditioned system, which influences the convergence of the used iterative method to a large extent.

At least for elliptic state problems leading to positive definite matrices $\mathbf{K}$ it is clear how to choose appropriate preconditioners $\hat{\mathbf{K}}$ for the state equation for the previous two preconditioners. On the other hand, it is by far more difficult how to choose preconditioners for the Schur-complement. One approach is to exploit mapping properties of the underlying pseudo-differential operator. This approach was used e.g. by TA'ASAN [147, 149, 146, 148] in the context of shape design for fluid dynamics. He used Fourier transformation to get the symbol of the Schur-complement and exploited this for preconditioning. A completely different approach was presented by BRAMBLE, PASCIAK, AND VASSILEVSKI [28]. They developed a way for the construction of efficient preconditioners of pseudo-differential operators of positive and negative order, based on multi-level techniques.

5.7 Examples and numerical results

In order to illustrate the previously described methods, we carry out some numerical experiments with our model problem described in Section 5.2. As the description there does not contain details on the observation as well as on the state equation, we will further restrict ourselves to two problem classes, namely the identification of a reaction coefficient and the identification of a conductivity.

5.7.1 The identification of a reaction coefficient

In our first example we want to identify the reaction coefficient $q \in H_0^1(\Omega)$ in the 1D potential equation

$$-u'' + qu = f \qquad \text{in } \Omega = (0,1), \tag{5.100}$$

$$u(0) = u(1) = 0 \tag{5.101}$$

The data z are an observation of $u \in L^2(\Omega)$, i.e. the observation operator E is the canonical embedding from $H^1(\Omega)$ into $L^2(\Omega)$. The right-hand side $f \in H^{-1}(\Omega)$ is given by

$$f(x) = \frac{1}{2} + \sin x, \qquad x \in \Omega,$$

the exact reaction coefficient $\hat{q} \in H_0^1(\Omega)$ by

$$\hat{q}(x) = x(1-x).$$

In other words, we consider the parameter identification problem

$$\frac{1}{2}\,\|u - z^\delta\|^2_{L^2(\Omega)} \to \min_{(u,q)\in H_0^1(\Omega)\times H_0^1(\Omega)} \tag{5.102}$$

subject to

$$\int_0^1 u'(x)v'(x) + q(x)u(x)v(x)\,\mathrm{d}x = \int_0^1 f(x)v(x)\,\mathrm{d}x \qquad \forall\, v \in H_0^1(\Omega) \tag{5.103}$$

where z^δ denotes a noisy approximation of the data z.

We use the LMSQP method, for which the KKT-system of the quadratic subproblem (5.31) looks as follows:

$$\begin{pmatrix} I_{L^2(\Omega)} & 0 & K_k^* \\ 0 & \beta_k I_{H_0^1(\Omega)} & L_k^* \\ K_k & L_k & 0 \end{pmatrix} \begin{pmatrix} u \\ q \\ \lambda \end{pmatrix} = \begin{pmatrix} (z^\delta - u_k) \\ 0 \\ -r_k \end{pmatrix}, \tag{5.104}$$

with

$$K_k : H_0^1(\Omega) \to H^{-1}(\Omega), \qquad \langle K_k u, v\rangle = \int_0^1 u'(x)v'(x) + q_k(x)u(x)v(x)\,\mathrm{d}x \qquad \forall\, v \in H_0^1(\Omega)$$

$$L_k : L^2(\Omega) \to H^{-1}(\Omega), \qquad \langle L_k q, v\rangle = \int_0^1 q(x)u_k(x)v(x)\,\mathrm{d}x \qquad \forall\, v \in H_0^1(\Omega)$$

and r_k denoting the residual of the state equation, i.e.

$$\langle r_k, v\rangle = \int_0^1 u_k'(x)v'(x) + q_k(x)u_k(x)v(x)\,\mathrm{d}x - \int_0^1 f(x)v(x)\,\mathrm{d}x \qquad \forall\, v \in H_0^1(\Omega).$$

As the state equation is linear in u and q in our case, the definition of K_k and L_k is straightforward.

For the numerical realization, Ω is discretized uniformly using linear finite elements. The approximations of the state variable u and the Lagrangian multiplier λ has m degrees of freedom, for approximating the parameter q we use n degrees of freedom. The noisy data z^δ are generated by solving the state equation on a fine grid using the exact reaction coefficient, restricting the fine grid solution to a coarser grid and finally adding some high-frequency perturbation as noise.

The simple structure of our example implies a rather simple structure of the KKT-submatrices of the Galerkin-LMSQP method, in particular in (5.80) $\mathbf{G}$ is an L^2-mass matrix, i.e.

$$\mathbf{G} = (\langle \phi_j, \phi_i\rangle_{L^2(\Omega)})_{i,j=1,\dots,m},$$

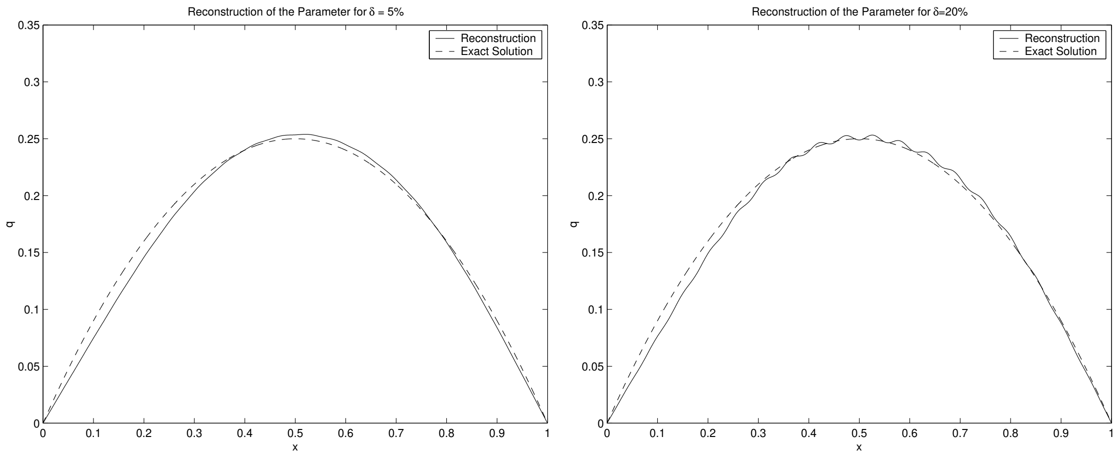

Figure 5.1: Reconstruction (solid) and exact solution (dashed) for noise level $\delta = 5\%$ (left) and $\delta = 20\%$ (right)

and $\mathbf{H}$ is an H^1-stiffness matrix, i.e.

$$\mathbf{H} = (\langle \sigma_j, \sigma_i \rangle_{H^1(\Omega)})_{i,j=1,\ldots,n}.$$

In the linearization of the state equation, $\mathbf{K}$ is defined via

$$\mathbf{K} = (\langle K_k \phi_j, \phi_i \rangle)_{i,j=1,\ldots,m}$$

and $\mathbf{L}$ via

$$\mathbf{L} = (\langle L_k \sigma_j, \phi_i \rangle)_{i=1,\ldots,m; j=1,\ldots,n} \ ,$$

where $\phi_i, i = 1, \ldots, m$ and $\sigma_j, j = 1, \ldots, m$ denote the according basis functions. Hardly any properties of the matrix $\mathbf{L}$ are known, except that $\mathbf{L}$ approximates a differential operator of order 0. We refer to (5.77), (5.78) for the definition of these matrices.

This problem is implemented in the software-system MATLAB. The KKT-system (5.80) is solved using a direct solver in this case, which is probably not the best choice with respect to the numerical effort for fine discretizations, but still yields reasonable results in our case. Figure 5.1 shows the results obtained with the LMSQP method for noise level $\delta = 5\%$ and $\delta = 20\%$. Surprisingly, the approximation is still reasonable even for a large noise level like $\delta = 20\%$, but the reconstruction is not as smooth as for $\delta = 5\%$ The corresponding evolutions of the error $\|q_k - \hat{q}\|_{H^1(\Omega)}$ and the residual $\|u_k - z^\delta\|_{L^2(\Omega)}$ are plotted in Figure 5.2. One observes that in both cases the error decreases up to some iteration index and then starts to increase again which is a typical phenomenon for inverse problems. That is why the iteration is not terminated according to the convergence criteria usually used in optimization, but due to an appropriate stopping rule (see e.g. Engl, Hanke, and Neubauer [50] for a general introduction). We used the so-called *discrepancy principle* as stopping criterion (the details can be found in Burger and Mühlhuber [33, 34]). We want to mention that the stopping index obtained from the discrepancy principle was always close to the iteration index, where the error is minimal.

The convergence of the LMSQP-method was compared to the Feasible-Path Levenberg-Marquardt method, introduced in Subsection 5.3.4. It turned out, that both methods lead to

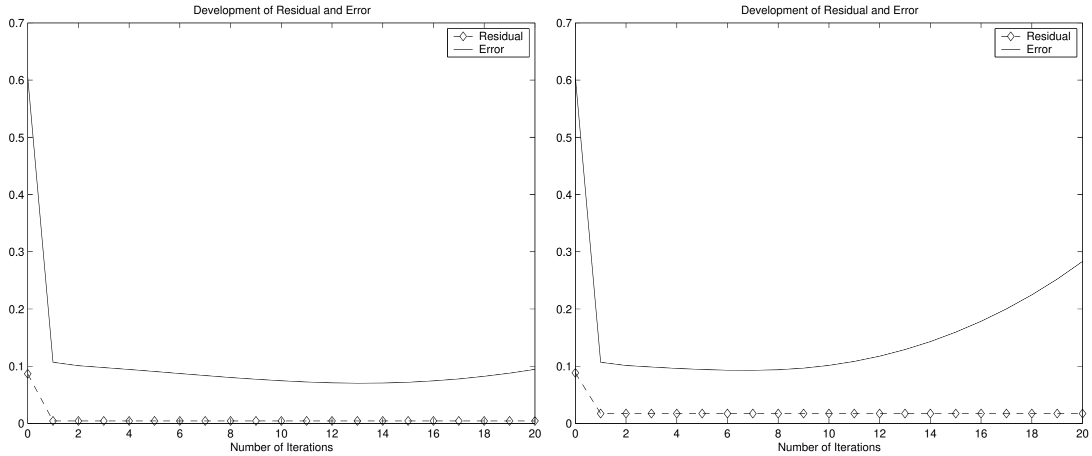

Figure 5.2: Development of the error $\|q_k - \hat{q}\|_{H^1(\Omega)}$ (solid) and the residual $\|u_k - z^\delta\|_{L^2(\Omega)}$ during the iteration for noise $\delta = 5\%$ (left) and $\delta = 20\%$ (right)

almost the same iteration sequence q_k. In particular, the number of iterations needed until the stopping rule is satisfied, is the same for both methods.

Finally, we compare the numerical efficiency of the LMSQP-method with feasible path approaches, namely the Feasible-Path Levenberg-Marquardt method (LM) (with the same Galerkin discretization as for LMSQP) and a Broyden-type variant of the LM-method (cf. KALTENBACHER [95] for further details).

For this sake we choose different discretization levels (fixed during the iteration) and measure the CPU-time needed for the LMSQP-method, until the stopping rule is satisfied (for fixed noise level δ). From the results shown in Table 5.1 and Figure 5.3 one observes that the LMSQP-method with simultaneous solution of the KKT-system outperforms the feasible-path approaches for all different discretizations. Since the LMSQP and the LM-method need the same number of outer iterations, the difference in the numerical effort is caused by the fact that the effort for the evaluation of the system matrix in the LM-method is significantly higher than evaluation and preconditioning of the system matrix in the simultaneous LMSQP-method. Obviously, the gain in the numerical effort for the evaluation of the system matrix increases with the number of discretization points, which explains the extremely large CPU-time for the LM-method at the finest discretization level ($m = 1601$). For small m and n, the Broyden-variant is much faster than the LM-method, which is again caused by the fact that the evaluation of the system matrix can be carried out efficiently. However, the number of iterations needed for the Broyden-type variant is much larger than for the other two methods, which use the full information about the derivatives.

These results also agree with the results presented in the previous chapter. There, we also used a Broyden update formula (strictly speaking, the BFGS update formula) for approximating the Hessian of the objective. But the behavior of the iteration was very similar: For a few design parameters we got a reasonable approximation of the Hessian, whereas for large design spaces the number of iterations dramatically increased. This led to the extremely large CPU-time reported in Table 4.3.

m	n	LMSQP	LM	Broyden
201	41	0.07	1.37	0.51
201	101	0.18	3.44	1.34
201	201	0.36	6.94	2.88
401	201	0.51	24.83	9.09
401	401	1.39	50.39	20.48
801	401	2.61	193.21	70.69
801	801	5.66	392.54	158.69
1601	801	7.91	1564.50	600.66
1601	1601	22.86	3144.40	1356.60

Table 5.1: CPU-time (in seconds) needed for the LMSQP-method, the LM-method and a Broyden-type variant of the LM-method

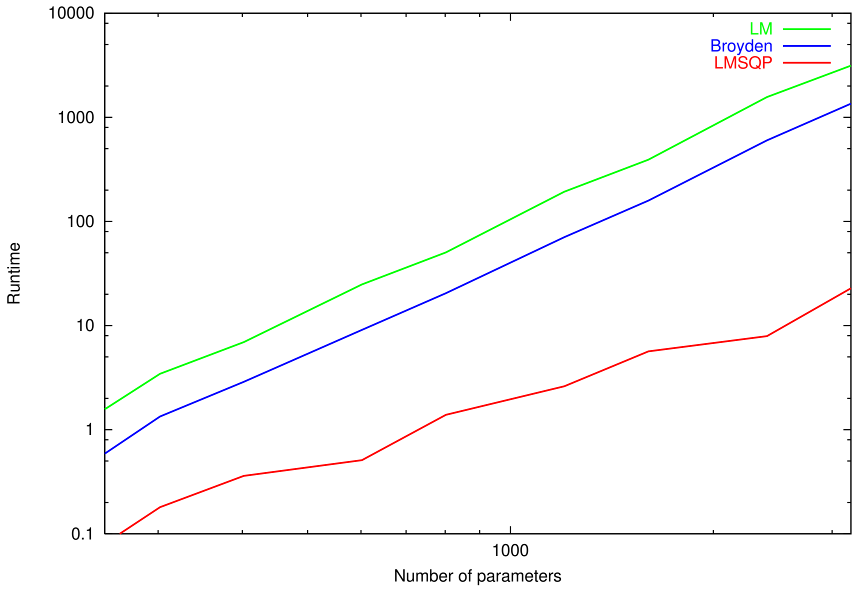

Figure 5.3: Comparison of the CPU-times for the LMSQP-method, the LM-Method and a Broyden-type variant of the LM-method

5.7.2 The identification of a conductivity

Our second numerical example is the identification of the conductivity $q \in L^\infty(\Omega) \subseteq L^2(\Omega)$ in

$$- \operatorname{div}(q \operatorname{grad} u) = f \qquad \text{in } \Omega, \tag{5.105}$$
$$u = 0 \qquad \text{on } \partial\Omega. \tag{5.106}$$

The data z are an observation of $u \in L^2(\Omega)$, i.e. the observation operator E is the canonical embedding of $H_0^1(\Omega)$ into $L^2(\Omega)$. The domain Ω is a ball in $\mathbb{R}^2$ with missing first quadrant (see also Figure 5.4), i.e., in radial coordinates

$$\Omega = \{(r\cos\theta, r\sin\theta) \mid r \in [0,1), \theta \in (\pi/2, 2\pi)\}\,. \tag{5.107}$$

The exact parameter to be reconstructed is $\hat{q} \equiv 1$, the right-hand side $f \in H^{-1}(\Omega)$ in (5.105) is given by

$$f = \frac{3\pi}{4}\big(3\pi\cos(\frac{3\pi}{2}r) + \frac{2}{r}\sin(\frac{3\pi}{2}r)\big) \qquad \text{with } r = \sqrt{x^2+y^2}.$$

The corresponding solution $\hat{u} \in H_0^1(\Omega)$ of the state equation is $\hat{u} = \cos(\frac{3\pi}{2}r)$. The noisy z^δ data are generated using the exact solution $\hat{u}$ perturbed by uniformly distributed random noise.

Summarizing, we consider the parameter identification problem

$$\frac{1}{2}\,\|u - z^\delta\|^2_{L^2\Omega} \to \min_{(u,q)\in H_0^1(\Omega)\times L^\infty(\Omega)} \tag{5.108}$$

subject to a weak formulation of the state problem (5.106).

We use the LMSQP-method for which the KKT-system of the quadratic subproblems looks as follows:

$$\begin{pmatrix} I_{L^2(\Omega)} & 0 & K_k^* \\ 0 & \beta_k I_{L^2(\Omega)} & L_k^* \\ K_k & L_k & 0 \end{pmatrix} \begin{pmatrix} u \\ q \\ \lambda \end{pmatrix} = \begin{pmatrix} (z^\delta - u_k) \\ 0 \\ -r_k \end{pmatrix}, \tag{5.109}$$

with

$$K_k : H_0^1(\Omega) \to H^{-1}(\Omega), \qquad \langle K_k u, v\rangle = \int_\Omega \langle q_k \operatorname{grad} u, \operatorname{grad} v\rangle \,\mathrm{d}x \qquad \forall\, v \in H_0^1(\Omega)$$
$$L_k : L^2(\Omega) \to H^{-1}(\Omega), \qquad \langle L_k q, v\rangle = \int_\Omega \langle q \operatorname{grad} u_k, \operatorname{grad} v\rangle \,\mathrm{d}x \qquad \forall\, v \in H_0^1(\Omega)$$

and r_k denoting the residual of the state equation, i.e.

$$\langle r_k, v\rangle = \int_\Omega \langle q_k \operatorname{grad} u_k, \operatorname{grad} v\rangle \,\mathrm{d}x - \int_\Omega f(x)v(x)\,\mathrm{d}x \qquad \forall\, v \in H_0^1(\Omega).$$

It is clear, that $L_k q$ does not exist for any $q \in L^2(\Omega)$, but only for $q \in L^\infty(\Omega)$. Thus, in the practical realization we have to introduce constraints on the parameter q. Usually, this can be done easily using a-priori information on the parameter, e.g. bounds on q.

For the discretization we used triangular finite elements with piecewise quadratic shape functions for the state u and the Lagrange parameter λ and piecewise constant shape functions

Level	dim q	dim u	avg QMR it	SQP it	time
2	92	215	200	9	8 sec
3	368	797	200	4	15 sec
4	1472	3065	180	5	77 sec
5	5888	12017	142	6	450 sec

Table 5.2: CPU-time and number of inner (QMR) and outer (SQP) iterations for exact data

Level	dim q	dim u	avg QMR it	SQP it	time	acc. time
2	92	215	200	9	8 sec	8 sec
3	368	797	200	4	15 sec	23 sec
4	1472	3065	175	2	24 sec	47 sec
5	5888	12017	80	1	47 sec	94 sec
6	23552	47585	121	1	425 sec	520 sec

Table 5.3: CPU-time per level, accumulated time and number of inner (QMR) and outer (SQP) iterations for exact data using a nested multi-level approach

for the parameter q. This implies a rather simple structure of KKT-sub-matrices in (5.80), in particular $\mathbf{G}$ and $\mathbf{H}$ are mass matrices with $\mathbf{H}$ being diagonal due to the choice of piecewise constant shape functions. For the detailed definition of $\mathbf{G}$, $\mathbf{H}$, $\mathbf{K}$, and $\mathbf{L}$, see (5.77), (5.78).

In order to ensure $q \in L^\infty(\Omega)$ we added box-constraints for $\mathbf{q}$ to the discretized optimization problem which are included using a barrier method (Nocedal and Wright [115]). As they never become active, we will not go into detail here.

The results were calculated using the finite element code FEPP (see Kuhn, Langer, and Schöberl [101]), developed at the Institute of Computational Mathematics of the University of Linz.

We want to mention that this identification problem is quite challenging not only due to the complicated geometry, but also due to the fact that q is not identifiable along a level line in the interior, where u attains an extremum. This does not destroy the theoretical identifiability results, because it is a set of Lebesgue-measure zero, but it can be expected to create numerical difficulties.

The KKT-system (5.80) was solved using a preconditioned QMR method with the block-factorization type preconditioner (5.98) with a multi-grid preconditioner $\hat{\mathbf{K}}$ and no preconditioning of the Schur-complement $\mathbf{S_c}$. Results for exact data can be found in Table 5.2. The good performance of the method with respect to both, CPU time and number of outer iterations can be observed clearly. Especially for problems with fine discretizations of the parameter q, this method can still be realized efficiently, while classical approaches do not yield results in reasonable time. A plot of the finite dimensional approximation of the parameter q can be found in Figure 5.4, from which one observes that the parameter is reconstructed very well except in a neighborhood of the level curve $\{\operatorname{grad} u = 0\}$.

Additional speed-up can be gained using a multi-level approach as described in Subsection 5.5.3. We used nested spaces for approximating q and u by subdividing each triangular element into four smaller elements, when refining the mesh. Table 5.3 presents results for this approach. It can be seen that on fine discretization levels one SQP step is sufficient for

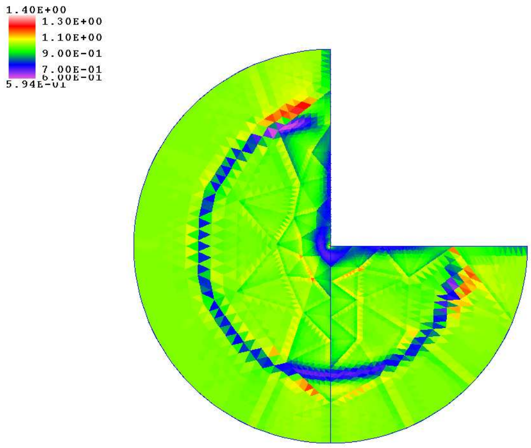

Figure 5.4: Parameter distribution for exact data at level 4, $q_{min} = 0.59$, $q_{max} = 1.4$

fulfilling the stopping criterion, which corresponds very well to the theoretical predictions (for details see Burger and Mühlhuber [34]. A comparison of the results to the ones in Table 5.2 shows that for fixed discretization level, the solution of the identification problem on level 5 is only slightly faster than the identification of q on level 6 (with about the fourfold number of parameters) using a multi-level approach (see also Figure 5.5).

A plot of the parameter can be found in Figure 5.6. Here the approximation of the parameter in the area where it can not identified is by far better than in the classical approach using only one discretization level (compare Figure 5.4). A possible explanation for this effect is the following: The influence of the level line $\{\operatorname{grad} u = 0\}$ where q can not be identified on the solution is smaller the coarser the discretization is. The prolongation from coarse levels to finer ones adds information to the region where the parameter is not identifiable from its surrounding region. As long as the parameter is smooth this helps to improve the quality of the numerical results where the parameter can not be identified.

5.8 Necessary changes for optimal design

In the following we want to discuss the necessary changes when changing from our model problem which is a parameter identification model to an optimal design problem.

First one has to adapt the optimization strategy. As one can not expect the Lagrangian multiplier to vanish (remember, our model problem was a least-squares problem, where we considered the special case for attainable data), the second order derivatives of the state equation can not be neglected. This implies that in the QP problem not a quadratic approximation of the objective, but of the Lagrangian (5.2) has to be used.

On the other hand, optimal design problems are usually not ill-posed. That is why, regularization procedures need not be used. Nevertheless, using the Levenberg-Marquardt modification of the objective can improve the convergence speed as it can also be interpreted as a trust region for the parameter. Summarizing, for optimal design problems either Method 1 or Method 2 has to be used.

Showing well-posedness of the QP problems for optimal design problems is usually by

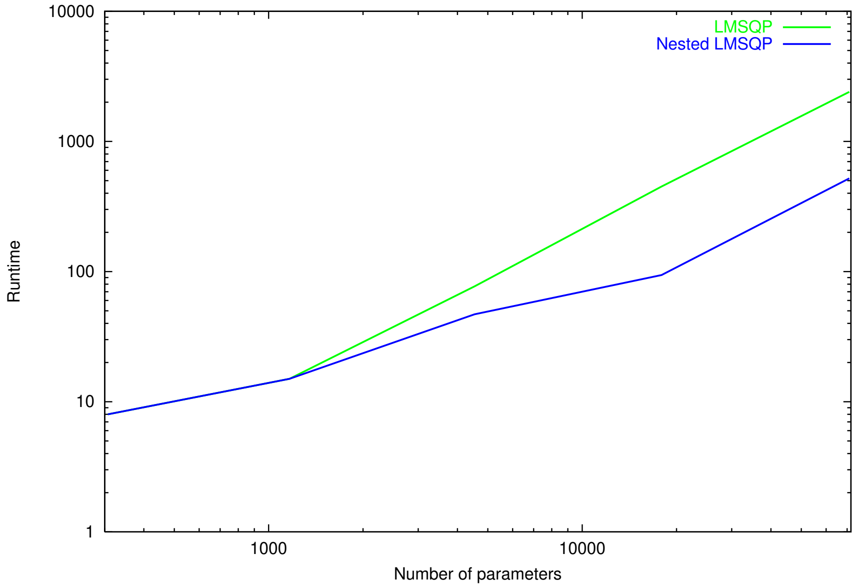

Figure 5.5: Comparison of the CPU-times for the LMSQP-method and its multi-level version using nested spaces

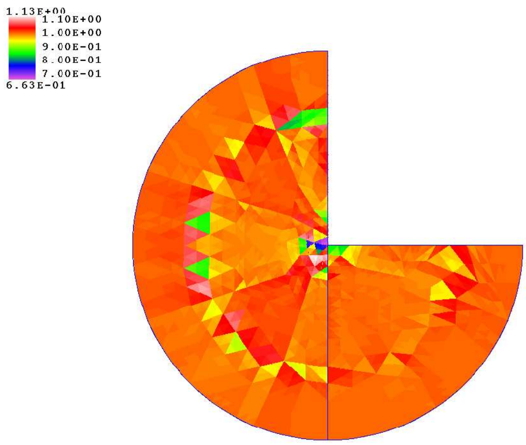

Figure 5.6: Parameter distribution for exact data at level 4 using a nested multi-level approach, $q_{min} = 0.66$, $q_{max} = 1.13$

far more complicated compared to our model problem. In our case, we had an explicit representation of the Hessian, as well as of the linearized state equation. Additionally, the situation was even more simplified by replacing the Hessian of the Lagrangian by the Hessian of the objective. Taking the Hessian of the Lagrangian, i.e. considering the IRSQP Method, makes the situation already slightly more complicated. In this situation we can still get well-posedness, but have to accept further restrictions on the regularization parameter (for details see BURGER AND MÜHLHUBER [33, 34]. Generalizing the objective can lead to situations, where it is hardly possible to show well-posedness because depending on the objective and the state equation only very little knowledge on the Hessian of the Lagrangian can be available.

Also the numerical realization of the algorithm is by far more difficult. For the evaluation of the objective on the QP problem first and second order derivatives are necessary. The first order derivatives can be calculated efficiently using automatic differentiation. Also the application of the Hessian, i.e. the evaluation of a Hessian times vector product can be done efficiently, i.e. the calculation time is proportional to the evaluation of the function itself. The calculation of the Hessian itself is usually by far more expensive (see e.g. GRIEWANK [67]. For an article on the use of AD in optimal design see e.g. KEYES, HOVLAND, MCINNES, AND SONYAMO [98].

Even more complicated than the evaluation of the KKT-matrix is its appropriate preconditioning. Although the preconditioning for our model is already quite difficult, here it is even worse. As usually the Hessian is not available (because of the computational effort) all preconditioning strategies needing matrix elements can not be applied. Also using mapping properties of the reduced Hessian is usually very difficult, if not even impossible. One possibility for preconditioning are strategies usually used for smaller optimization problems. E.g. the BFGS-update formula can provide you with an approximation of the reduced Hessian which can be used in combination with the preconditioners of type (5.98) or (5.99). Usually not the standard BFGS formula, but limited memory variants are used for maintaining sparsity (NOCEDAL AND WRIGHT [115]) which makes the efficient application of the preconditioners possible. For an example using this approach see BIROS AND GHATTAS [16, 17].

Chapter 6

Some Remarks on the Software Design

6.1 Introduction

The goal of our optimal design library is to provide a flexible frame for optimal design problems. During the development of the here presented design, we put strong emphasis on a flexible and easy administrable tool and not on the highest possible efficiency. Nevertheless, the design is also efficient, as could be seen in the results presented in Chapter 4 and Chapter 5. Additionally, we tried to develop a frame which is more or less independent of the used optimizer and the finite element package used. That is why, one of the main principles during the development was a strict splitting of the optimizer on the one hand and the realization of the optimization problem on the other. Exchanging the optimizer by a different one (e.g. an optimizer written in Fortran) induces hardly any changes on the implementation of the optimal design problem (objective, constraints, state equation). Also different packages for solving the state problem can be easily integrated. Although we restrict ourselves to an FE solver for the state equation in the following considerations, using finite differences, the finite volume method or any other method for calculating an approximate solution of our state equation would be appropriate. A predecessor of the here presented software design was presented by Kuhn, Lukáš, and Mühlhuber [102].

6.2 The optimization modules

In our code, the optimizer is only based on a linear algebra package, which provides vectors, matrices, etc. On top of these basic linear algebra data types, we have built different optimization strategies:

- An optimizer for unconstrained optimization problems, which is based on a quasi-Newton method using a BFGS update formula for approximating the Hessian of the objective. To achieve global convergence we included a line search method. As the optimizer is based on dense matrices, it is only suitable for small optimization problems with rather few design parameters.
- A QP optimizer for linearly constrained optimization problems with quadratic objective. For this optimizer we additionally assumed that the Hessian of the objective is positive

definite. The optimizer is based on dense matrices and therefore only suitable for small optimization problems with rather few design parameters. The QP module is available in two different implementations: one based on null-space methods and one based on a range-space approach. Both implementations use formulas for updating a basis of the null-space of the active constraints which reduce the effort in each iteration when adding or deleting a constraint from the active index set. The main application of this optimizer is the calculation of the search direction in an SQP method.

- An SQP optimizer for small nonlinearly constrained optimization problems. This module is based on dense matrices and therefore only suitable for optimization problems with rather few design parameters (c.f. Chapter 4). To get a search direction we solve a QP problems by one of the optimization modules described in the previous item. For globalization we use a line search method. The Hessian of the Lagrangian is approximated by a quasi-Newton method using a modified BFGS update formula following POWELL [122].
- For the approach presented in Chapter 5 we implemented an optimizer which works in the product space $(\mathbf{u}, \mathbf{q})$. Only $\mathbf{q}$ is represented by a vector, $\mathbf{u}$ uses an abstract base class vector type to be flexible with respect to the representation of the state solution. At the moment the state-equation is the only supported constraint but a generalization to support also constraints on $\mathbf{q}$ or $\mathbf{u}$ or on both variables would be easily possible. The Hessian of the Lagrangian as well as the partial derivatives of the state equation need not be assembled, only corresponding matrix-vector operations need to be provided. The QP subproblem is solved by an iterative solver (at the moment a QMR module, c.f. FREUND AND NACHTIGAL [56]) where appropriate preconditioners have to be provided by the user. This enables us to solve also large scale optimal design problems with many design and state parameters as long as suitable preconditioners for the QP subproblem are available.

Each of these optimization modules was implemented in C++ and uses heavily the concepts of operator overloading and virtual inheritance. In order to obtain good performance we use a sophisticated memory management to eliminate temporary object creation.

In the following section we describe the general frame for implementing the optimal design problem.

6.3 The optimal design problem

The frame for the efficient implementation of an optimal design problem presented here encapsulates the communication between the optimizer and the finite element code. Both, the state equation and the objective are treated as abstract objects which enables us to be independent of the specific type of problem. Thus, different types of optimal design problems can be realized in this frame (e.g. shape or topology optimization), but also different classes of state equations can be easily integrated. Especially, it is not necessary to have state problems of elliptic type or linear state problems. In this section we will also show how to realize all methods which are needed to implement the algorithms proposed in Chapter 4 and Chapter 5.

The evaluation of an objective like the one used in Chapter 4 can be split into several subtasks. A flow graph of the function evaluation for this class of objective can be found in Figure 6.1. Motivated by this figure, we introduce the following three objects to realize a

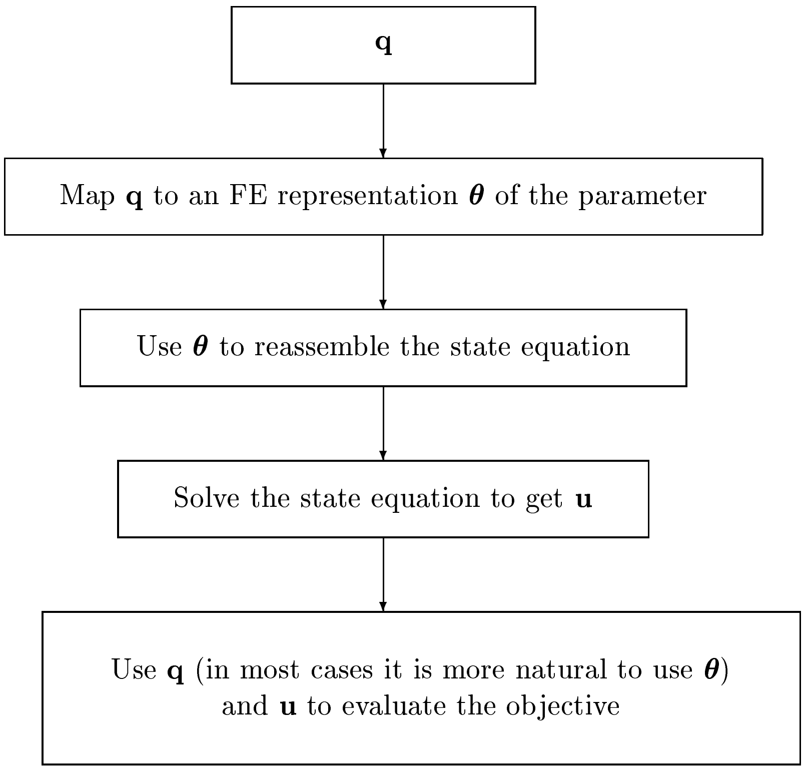

Figure 6.1: Flow graph of a typical function evaluation

function evaluation:

- a *ParameterMap*,
- a *StateConstraint* and
- a *ProductSpaceFunction*.

In the following, we will discuss these objects in more detail and show which functionality is necessary for each of these objects. Additionally, we will discuss how a direct or an adjoint method can be realized, as well how to integrate AD into this framework.

6.3.1 *ParameterMap*

This object is the most important communication layer between the optimizer and the finite element code. As could be seen in Figure 6.1 it maps the parameter $\mathbf{q}$ handled by the optimizer to a representation $\boldsymbol{\theta}$ of the parameter in the FE code. This representation can be of very different nature for different problems:

- When considering a sizing problem, $\boldsymbol{\theta}$ will be a representation of the thickness. In topology optimization $\boldsymbol{\theta}$ represents the density of the material. In the parameter identification problem considered in Chapter 5 it was the unknown conductivity to be identified. In all these cases, the mapping is more or less straight-forward.
- In an optimal control problem θ is the representation of the control in the FE code (e.g. as a piecewise constant or linear function).
- For shape optimization $\mathbf{q}$ represents a parameterization of the computational domain. In this situation, $\boldsymbol{\theta}$ could be a representation of the nodes of the FE mesh, respectively their coordinates of the domain corresponding to the parameter $\mathbf{q}$. Thus for getting $\boldsymbol{\theta}$ it is necessary to generate a geometry model corresponding to $\mathbf{q}$ by using e.g. a parametric CAD modeler, and then to deform the mesh of the reference geometry to get a mesh of the current geometry (see also the remarks in Section 1.3).

The *StateConstraint* and the *ProductSpaceObjective* use both the *ParameterMap* for their communication with the optimizer. As they calculate derivatives only with respect to $\boldsymbol{\theta}$ the *ParameterMap* also has to provide functionality to map gradients and other derivatives with respect to $\boldsymbol{\theta}$ to the corresponding derivatives with respect to $\mathbf{q}$.

6.3.2 *StateConstraint*

The *StateConstraint* manages all accesses to the state equation and is therefore an encapsulation of functionality of the FE code, i.e. it represents

$$\mathbf{e}(\mathbf{u}, \mathbf{q}) = 0.$$

Usually a *StateConstraint* uses a *ParameterMap* for the communication to the optimizer i.e. it accesses only the output $\boldsymbol{\theta}$ of the *ParameterMap* and not the parameter $\mathbf{q}$ directly.

For a product space approach as in Chapter 5 the following functionality is necessary:

- Evaluate $\mathbf{e}$ for given state $\mathbf{u}_0$ and parameter $\mathbf{q}_0$.

- Evaluate the gradient with respect to $\mathbf{u}$ and $\mathbf{q}$ of

$$\mathbf{v}^T \mathbf{e}(\mathbf{u}_0, \mathbf{q}_0)$$

  for given $\mathbf{v}$, $\mathbf{u}_0$, $\mathbf{q}_0$.

- Evaluate the linearization of $\mathbf{e}$, i.e. especially evaluate

$$\frac{\partial \mathbf{e}}{\partial \mathbf{u}}(\mathbf{u}_0, \mathbf{q}_0)\, \Delta\mathbf{u} + \frac{\partial \mathbf{e}}{\partial \mathbf{q}}(\mathbf{u}_0, \mathbf{q}_0)\, \Delta\mathbf{q}.$$

  for given $\mathbf{u}_0$, $\mathbf{q}_0$, $\Delta\mathbf{u}$, $\Delta\mathbf{q}$.

- Assemble the Jacobian of $\mathbf{e}$ if possible with suitable effort, i.e.

$$\frac{\partial \mathbf{e}}{\partial \mathbf{u}} \quad \text{and} \quad \frac{\partial \mathbf{e}}{\partial \mathbf{q}}.$$

For reduced SQP approaches (as in Chapter 4) we need additional functionality. Especially, we have to linearize the state equation and to solve the linearized equation and its adjoint. Usually, the linearized problem is represented as

$$\Delta\mathbf{u} = -\Big(\frac{\partial \mathbf{e}}{\partial \mathbf{u}}(\mathbf{u}_0, \mathbf{q}_0)\Big)^{-1} \mathbf{e}(\mathbf{u}_0.\mathbf{q}_0) - \Big(\frac{\partial \mathbf{e}}{\partial \mathbf{u}}(\mathbf{u}_0, \mathbf{q}_0)\Big)^{-1} \frac{\partial \mathbf{e}}{\partial \mathbf{q}}(\mathbf{u}_0, \mathbf{q}_0)\, \Delta\mathbf{q},$$

where $(\mathbf{u}_0, \mathbf{q}_0)$ denotes the linearization point. Summarizing we need:

- Linearize the state equation and define the operators

$$\mathbf{R} = -\Big(\frac{\partial \mathbf{e}}{\partial \mathbf{u}}(\mathbf{u}_0, \mathbf{q}_0)\Big)^{-1} \qquad \text{and} \qquad \mathbf{S} = \frac{\partial \mathbf{e}}{\partial \mathbf{q}}(\mathbf{u}_0, \mathbf{q}_0).$$

- Evaluate

$$\Delta\mathbf{u} = \mathbf{R}\,\mathbf{S}\,\Delta\mathbf{q}$$

  for given $\Delta\mathbf{q}$ which involves the solution of the linearized state problem with a given right-hand side.

- Evaluate

$$\Delta\mathbf{q} = \mathbf{S}^T\,\mathbf{R}^T\,\Delta\mathbf{u}$$

  for given $\Delta\mathbf{u}$ which involves the solution of the adjoint of the linearized state problem for a given right-hand side.

- Evaluate

$$\Delta\boldsymbol{\lambda} = \mathbf{R}^T\,\Delta\mathbf{u}$$

  for given $\Delta\mathbf{u}$ which again involves the solution of the adjoint linearized state problem for a given right-hand side.

The direct and the adjoint method can also be realized with this functionality (c.f. (4.21), (4.22), (4.23)) as long as partial derivatives of the objective with respect to state and design parameter are available.

The design of the *StateConstraint* presented here is not only applicable to PDEs of elliptic type but can be easily used also for time-dependent PDEs.

6.3.3 *ProductSpaceFunction*

The *ProductSpaceFunction* encapsulates the evaluation of the objective and its derivatives. Additionally it can be used to realize constraints depending on state and design parameters.

As the name implies, a *ProductSpaceFunction* realizes a function depending on design **and** state parameters. Similar to the *StateConstraint* it uses a *ParameterMap* for the communication to the optimizer and accesses only the FE representation $\boldsymbol{\theta}$ of the parameter.

For applying the optimization procedures presented in Chapter 4 and Chapter 5 it suffices to make the following functionality available:

- Evaluate the function for given state and design parameter.
- Evaluate the first derivative of the function for given state and design parameter.
- Apply the second order derivative to a given vector.
- Assemble the Hessian matrix if possible with suitable effort.

The presented frame makes it also easy to incorporate AD. As explained in Chapter 4 it does not make sense to apply AD in a black-box manner to an optimal design problem. It is better to use hand-coded parts for solving the linearized state problems (in direct or adjoint fashion) and apply AD only to get partial derivative of the objective itself (in the terminology of this section to use hand-coded derivatives for the *StateConstraint* and AD for the *ProductSpaceFunction*). This was also suggested as hybrid method in Subsection 4.4.4. The here presented design supports the implementation of such a splitting to a large extent. We tried to keep the information local which makes it easy to realize such a combination of different differentiation strategies.

Bibliography

[1] Robert A. Adams, *Sobolev spaces*, Pure and Applied Mathematics, vol. 65, Academic Press, New York - San Francisco - London, 1975.

[2] Narayana R. Aluru and James White, *Direct Newton finite-element / boundary-element technique for micro-electro-mechanical analysis*, Solid-State Sensor and Actuator Workshop (Hilton Head, South Carolina), 1996, pp. 54 – 57.

[3] Owe Axelsson, *Iterative solution methods*, Cambridge University Press, Cambridge, 1994.

[4] Ivo Babuska and Theolanis Strouboulis, *The finite element method and its reliability*, Numerical Mathematics and Scientific Computation, Oxford University Press, Oxford - New York, 2001.

[5] H. T. Banks and Karl Kunisch, *Estimation techniques for distributed parameter systems*, Systems & Control: Foundations & Applications, vol. 1, Birkhäuser, Basel - Boston - Berlin, 1989.

[6] Wolf Bartelheimer, *Ein Entwurfsverfahren für Tragflügel in transsonischer Strömung*, DLR-Forschungsbericht 96-30, Deutsche Forschungsanstalt für Luft- und Raumfahrt (DLR), Braunschweig, 1996, ISSN 0939 - 2963.

[7] Klaus-Jürgen Bathe, *Finite element procedures*, Prentice Hall, 1996.

[8] Astrid Battermann and Matthias Heinkenschloss, *Preconditioners for Karush-Kuhn-Tucker matrices arising in the optimal control of distributed systems*, Optimal Control of Partial Differential Equations (Vorau 1997) (Wolfgang Desch, Franz Kappel, and Karl Kunisch, eds.), Birkhäuser, Basel - Bosten - Berlin, 1998, pp. 15 – 32.

[9] Astrid Battermann and Ekkehard W. Sachs, *Block preconditioners for KKT systems in PDE-governed optimal control problems*, Fast solution of discretized optimization problems (Workshop held at the Weierstrass Institute for Applied Analysis and Stochastics, Berlin, Germany) (Heinz-Karl Hoffmann, Ronald H. W. Hoppe, and Volker Schulz, eds.), ISNM, Internat. Series Numer. Math., vol. 138, Birkhäuser, Basel, 2001, pp. 1 – 18.

[10] Roland Becker, Hartmut Kapp, and Rolf Rannacher, *Adaptive finite element methods for optimal control of partial differential equations: Basic concepts*, SIAM J. Control Optim. **39** (2000), no. 1, 113 – 132.

[11] ———, *Adaptive finite element methods for optimization problems*, Proceedings of the 18th Dundee biennial conference (University of Dundee, GB) (D. F. Griffiths et al., ed.), CRC Res. Notes Math., no. 420, Chapman & Hall / CRC, Boca Raton, FL, 2000, pp. 21 – 42.

[12] Martin P. Bendsøe, *Optimal shape design as a material distribution problem*, Struct. Multidisc. Optim. **1** (1989), 193 – 202.

[13] Martin P. Bendsøe, *Optimization of structural topology, shape and material*, Springer, 1995.

[14] Claus Bendtsen and Ole Stauning, *FADBAD, a flexible C++ package for automatic differentiation*, Tech. Report IMM-REP-1996-17, Technical University of Denmark, IMM, Department of Mathematical Modeling, Lyngby, Denmark, 1996.

[15] Martin Berz, Christian Bischof, George Corliss, and Andreas Griewank (eds.), *Computational differentiation: Techniques, applications, and tools*, Philadelphia, Penn., SIAM, 1996.

[16] George Biros and Omar Ghattas, *Parallel Lagrange-Newton-Krylov-Schwarz methods for PDE-constrained optimization problems. Part 1: The Krylov-Schur solver*, Preprint, Carnegie-Mellon University, 2000.

[17] ———, *Parallel Lagrange-Newton-Krylov-Schwarz methods for PDE-constrained optimization problems. Part 2: The Lagrange-Newton solver and its application to optimal control of steady viscous flow*, Preprint, Carnegie-Mellon University, 2001.

[18] Christian H. Bischof, Alan Carle, George F. Corliss, Andreas Griewank, and Paul Hovland, *ADIFOR: Generating derivative code from Fortran programs*, Scientific Programming **1** (1992), 11 – 29.

[19] Christian H. Bischof, Alan Carle, Peyvand M. Khademi, and Andrew Mauer, *The ADIFOR 2.0 system for the automatic differentiation of Fortran 77 programs*, Comput. Sci. Engrg. **3** (1996), no. 3, 18 – 32.

[20] Christian H. Bischof, Lucas Roh, and Andrew Mauer, *ADIC — An extensible automatic differentiation tool for ANSI-C*, Software–Practice and Experience **27** (1997), no. 12, 1427 – 1456.

[21] Paul T. Boggs and Jon W. Tolle, *Sequential quadratic programming*, Acta Numerica **4** (1995), 1 – 51.

[22] Thomas Borrvall, *Computational topology optimization of elastic continua by design restrictions*, Linköping Studies in Science and Technology, Thesis 848, Division of Mechanics, Department of Mechanical Engineering, Linköping University, Sweden, 2000.

[23] Thomas Borrvall and Joakim Petersson, *Large scale topology optimization in 3D using parallel computing*, Tech. Report LiTH-IKP-R-1130, Department of Mechanical Engineering, Linköping University, 2000.

[24] ———, *Topology optimization using regularized intermediate density control*, Tech. Report LiTH-IKP-R-1086, Department of Mechanical Engineering, Linköping University, 2000, To appear in Comput. Methods Appl. Mech. Engrg.

[25] Blaise Bourdin, *Filters in topology optimization*, DCAMM Report 627, Technical University of Denmark, 1999.

[26] Dietrich Braess, *Finite Elemente – Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*, second ed., Springer, Berlin - Heidelberg - New York, 1997.

[27] James H. Bramble, Joseph E. Pasciak, and Apostol T. Vassilev, *Analysis of the inexact Uzawa algorithm for saddle point problems*, SIAM J. Numer. Anal. **34** (1997), 1072 – 1092.

[28] James H. Bramble, Joseph E. Pasciak, and Panayot S. Vassilevski, *Computational scales of Sobolev norms with application to preconditioning*, Math. Comp. **69** (2000), 463 – 480.

[29] Susanne C. Brenner and L. Ridgway Scott, *The mathematical theory of finite element methods*, Texts in Applied Mathematics, vol. 15, Springer, New York - Berlin - Heidelberg, 1994.

[30] Franco Brezzi, *On the existence, uniqueness and approximation of saddle-point problems arising from Lagrangian multipliers*, RAIRO Anal. Numér. **8** (1974), 129 – 151.

[31] Franco Brezzi and Michel Fortin, *Mixed and hybrid finite element methods*, Springer Series in Computational Mathematics, vol. 15, Springer, New York - Berlin - Heidelberg, 1991.

[32] Tyler E. Bruns and Daniel A. Tortorelli, *Topology optimization of nonlinear elastic structures and compliant mechanisms*, Report, Department of Mechanical and Industrial Engineering, University of Illinois, Urbana-Champaign, 1999.

[33] Martin Burger and Wolfram Mühlhuber, *Iterative regularization of parameter identification problems by SQP methods*, SFB-Report 01-18, SFB F013, University of Linz, Austria, May 2001, submitted to Inverse Problems.

[34] ———, *Numerical approximation of an SQP-type method for parameter identification*, SFB-Report 01-19, SFB F013, University of Linz, Austria, May 2001, submitted to SIAM J. Numer. Anal.

[35] Jean Céa and Kazimierz Malanowski, *An example of a max-min problem in partial differential equations*, SIAM J. Control Optim. **8** (1970), 305 – 316.

[36] Isabelle Charpentier, *Checkpointing schemes for adjoint codes: Applications to the meteorological model meso-NH*, SIAM J. Sci. Comput. **22** (2001), no. 6, 2135 – 2151.

[37] Guy Chavent and Karl Kunisch, *On weakly nonlinear inverse problems*, SIAM J. Appl. Math. **56** (1996), 542 – 572.

[38] Goong Chen and Jianxin Zhou, *Boundary element methods*, Computational Mathematics and Applications, Academic Press, Harcourt Brace Jovanovich, London - San Diego - New York, 1992.

[39] Philippe G. Ciarlet, *The finite element method for elliptic problems*, second ed., A Series of Comprehensive Studies in Mathematics, North-Holland Publishing Company, Amsterdam - New York - Oxford-Tokyo, 1987.

[40] ______, *Mathematical elasticity: Three-dimensional elasticity*, Studies in Mathematics and Its Applications, vol. 20, North-Holland Publishing Company, Amsterdam - New York - Oxford - Tokyo, 1994.

[41] Andrew R. Conn, Nicolas I. M. Gould, and Philippe L. Toint, *Trust-region methods*, MPS-SIAM Series on Optimization, vol. 1, SIAM, Philadelphia, 2000.

[42] George Corliss, Christéle Faure, Andreas Griewank, Laurent Hascoët, and Uwe Naumann (eds.), *Automatic differentiation of algorithms: From simulation to optimization*, New York, Springer, 2001.

[43] Marc Dambrine and Michel Pierre, *About stability of equilibrium shapes*, Math. Modelling Numer. Anal. **34** (2000), no. 8, 811 – 834.

[44] Michel C. Delfour and Jean-Paul Zolésio, *Shapes and geometries: Analysis, differential calculus, and optimization*, Advances in Design and Control, vol. 4, SIAM, 2001.

[45] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM J. Matrix Anal. Appl. **20** (1999), no. 3, 720 – 755.

[46] Peter Deuflhard and Andreas Hohmann, *Numerische Mathematik I*, second ed., Walter de Gruyter & Co., Berlin - New York, 1993.

[47] Iain S. Duff, Albert M. Erisman, and John K. Reid, *Direct methods for sparse matrices*, Monographs on Numerical Analysis, Clarendon Press, Oxford, 1986.

[48] Wolfgang Egartner and Volker Schulz, *Partially reduced SQP methods for optimal turbine and compressor blade design*, ENUMATH 97, Proceedings of the Second European Conference on Numerical Mathematics and Advanced Applications (Hans Georg Bock, Guido Kanschat, Rolf Rannacher, Franco Brezzi, Roland Glowinski, Yuri A. Kuznetsov, and Jacques Periaux, eds.), World Scientific Publishers, 1998, pp. 286 – 293.

[49] Howard C. Elman and Gene H. Golub, *Inexact and preconditioned Uzawa algorithms for saddle point problems*, SIAM J. Numer. Anal. **31** (1994), 1645 – 1661.

[50] Heinz W. Engl, Martin Hanke, and Andreas Neubauer, *Regularization of inverse problems*, Mathematics and Its Applications, vol. 375, Kluwer Academic Publishers, Dordrecht - Boston - London, 1996.

[51] Heinz W. Engl, Karl Kunisch, and Andreas Neubauer, *Convergence rates for Tikhonov regularization on nonlinear ill-posed problems*, Inverse Problems **5** (1989), 523 – 540.

[52] Heinz W. Engl and Otmar Scherzer, *Convergence rate results for iterative methods for solving nonlinear ill-posed problems*, Solution Methods for Inverse Problems (David Colton, Heinz W. Engl, Joyce R. McLaughlin, Alfred K. Louis, and William Rundell, eds.), Springer, Vienna - New York, 2000.

[53] Hans A. Eschenauer and Niels Olhoff, *Topology optimization of continuum structures: A review*, ASME Appl. Mech. Rev. **54** (2001), no. 4, 331 – 390.

[54] Lawrence C. Evans, *Partial differential equations*, Graduate Studies in Mathematics, vol. 19, American Mathematical Society, Providence, Rhode Island, 1998.

[55] Robert Fletcher, *Practical methods of optimization*, vol. 2, John Wiley & Sons, New York, 1981.

[56] Roland W. Freund and Noël M. Nachtigal, *QMR: a quasi-minimal residual method for non-Hermitian linear systems*, Numer. Math. **60** (1991), 315 – 339.

[57] Alan George and Joseph W. H. Liu, *Computer solution of large sparse positive definite systems*, Prentice-Hall Series in Computational Mathematics, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[58] Ralf Giering, *Tangent linear and adjoint model compiler, USER manual*, 1997, available under `http://puddle.mit.edu/~ralf/tamc/`.

[59] Ralf Giering and Thomas Kaminski, *Recipes for adjoint code construction*, Tech. Report 212, Max-Planck Institut für Meteorologie Hamburg, 1996.

[60] ———, *Recipes for adjoint code construction*, ACM Trans. Math. Software **24** (1998), no. 4, 437 – 474.

[61] ———, *On the performance of derivative code generated by TAMC*, Tech. report, Max-Planck Institut für Meteorologie, Hamburg, Germany, 2000, submitted to Optim. Methods Softw.

[62] Philip E. Gill, Walter Murray, Michael Saunders, and Margaret H. Wright, *Some theoretical properties of an augmented Lagrangian merit function*, Advances in Optimization and Parallel Computing (P. M. Pardalos, ed.), North-Holland, 1992, pp. 101 – 128.

[63] Philip E. Gill, Walter Murray, and Margaret H. Wright, *Practical optimization*, Academic Press, Inc., London - San Diego - New York, 1981.

[64] Vivette Girault and Pierre-Arnoud Raviart, *Finite element methods for the Navier-Stokes equations – theory and algorithms*, Springer, Berlin - Heidelberg - New York, 1986.

[65] Donald Goldfarb and A. Idnani, *A numerically stable dual method for solving strictly convex quadratic programs*, Math. Programming **27** (1983), 1 – 33.

[66] Andreas Griewank, *Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation*, Optim. Methods Softw. **1** (1992), 35 – 54.

[67] ———, *Evaluating derivatives: Principles and techniques of algorithmic differentiation*, Frontiers in Applied Mathematics, vol. 19, SIAM, Philadelphia, 2000.

[68] Andreas Griewank and George F. Corliss (eds.), *Automatic differentiation of algorithms: Theory, implementation, and application*, Philadelphia, Penn., SIAM, 1991.

[69] Andreas Griewank, David Juedes, Hristo Mitev, Jean Utke, Olaf Vogel, and Andrea Walther, *ADOL-C: A package for the automatic differentiation of algorithms written in C/C++*, Tech. report, Technical University of Dresden, Institute of Scientific Computing and Institute of Geometry, 1999, Updated version of the paper published in [70].

[70] Andreas Griewank, David Juedes, and Jean Utke, *ADOL-C: A package for the automatic differentiation of algorithms written in C/C++*, ACM Trans. Math. Software **22** (1996), no. 2, 131 – 167.

[71] Andreas Griewank and Andrea Walther, *Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation*, ACM Trans. Math. Software **26** (2000), no. 1, 19 – 45.

[72] Christian Grossmann and Hans-Georg Roos, *Numerik partieller Differentialgleichungen*, Teubner, Stuttgart, 1992.

[73] Gundolf Haase and Ewald Lindner, *Advanced solving techniques in optimization of machine components*, Comput. Assist. Mech. Engrg. Sci. **6** (1999), no. 3, 337 – 343.

[74] Eldad Haber and Uri Ascher, *Preconditioned all-at-once methods for large, sparse parameter estimation problems*, Tech. report, University of British-Columbia, Vancouver, Canada, July 2001.

[75] Robert B. Haber, Chandrashekhar S. Jog, and Martin P. Bendsøe, *A new approach to variable-topology shape design using a constraint on perimeter*, Struct. Multidisc. Optim. **11** (1996), 1 – 12.

[76] Wolfgang Hackbusch, *Multigrid methods and applications*, Springer, Berlin - Heidelberg - New York, 1985.

[77] ______, *Iterative Lösung großer schwachbesetzter Gleichunssysteme*, Teubner Studienbücher Mathematik, B. G. Teubner, Stuttgart, 1991.

[78] Shih-Ping Han, *A globally convergent method for nonlinear programming*, J. Optim. Theory Appl. **22** (1977), 297 – 309.

[79] Martin Hanke, *A regularizing Levenberg-Marquardt scheme with applications to inverse groundwater filtration problems*, Inverse Problems **13** (1997), 79 – 95.

[80] Martin Hanke, Andreas Neubauer, and Otmar Scherzer, *A convergence analysis of the Landweber iteration for nonlinear ill-posed problems*, Numer. Math. **72** (1995), 21 – 37.

[81] Jaroslav Haslinger and Pekka Neittaanmäki, *Finite element approximation for optimal shape design: Theory and applications*, John Wiley & Sons Ltd., Chinchester, 1988.

[82] Matthias Heinkenschloss, *The numerical solution of a control problem governed by a phase field model*, Optim. Methods Softw. **7** (1997), 211 – 263.

[83] Bernd Heinrich, *Finite difference methods on irregular networks. a generalized approach to second order elliptic problems*, International Series of Numerical Mathematics, vol. 82, Birkhäuser, Basel, 1987.

[84] Bodo Heise, *Nonlinear field calculations with multigrid Newton methods*, Impact Comput. Sci. Engrg. **5** (1993), 75 – 110.

[85] Magnus R. Hestenes and Eduard Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards **49** (1952), 409 – 436.

[86] Michael Hintermüller and Wolfgang Ring, *A level set approach for the solution of a state constrained optimal control problem*, Tech. report, Institute of Mathematics, University of Graz, Austria, February 2002.

[87] Ronald W. Hoppe, Svetozara Petrova, and Volker Schulz, *3D structural optimization in electromagnetics*, Proceedings of Domain Decomposition Methods and Applications (Lyon, Oct. 9-12, 2000) (M. Garbey et al., ed.), 2001.

[88] ______, *A primal-dual Newton-type interior-point method for topology optimization*, J. Optim. Theory Appl. **114** (2002), no. 3, to appear.

[89] Josef Hoschek and Dieter Lasser, *Grundlagen der geometrischen Datenverarbeitung*, B. G. Teubner, Stuttgart, 1989.

[90] Thomas J. R. Hughes, *The finite element method: Linear static and dynamic finite element analysis*, Dover Publications, 2000.

[91] Nathan Ida and Joao P. A. Bastos, *Electromagnetics and calculation of fields*, Springer, 1997.

[92] Michael Jung and Ulrich Langer, *Applications of multilevel methods to practical problems*, Surveys Math. Indust. (1991), 217 – 257.

[93] ______, *Methode der finiten Elemente für Ingenieure – Eine Einführung in die numerischen Grundlagen und Computersimulation*, Teubner, Stuttgart - Leipzig - Wiesbaden, 2001.

[94] Barbara Kaltenbacher, *Some Newton-type methods for the regularization of nonlinear ill-posed problems*, Inverse Problems **13** (1997), 729 – 753.

[95] ______, *On Broyden's method for the regularization of non-linear ill-posed problems*, Numer. Funct. Anal. Optim. **19** (1998), 807 – 833.

[96] Manfred Kaltenbacher, Hermann Landes, Reinhard Lerch, and Franz Lindinger, *A finite-element / boundary element method for the simulation of coupled electrostatic-mechanical systems*, J. Physique III **7** (1997), 1975 – 1982.

[97] Manfred Kaltenbacher, Hermann Landes, Kurt Niederer, and Reinhard Lerch, *3D simulation of controlled micro-machined capacitive ultrasound transducers*, Proceedings of the IEEE Ultrasonic Symposium (Lake Tahoe), 1999, accepted for publication.

[98] David E. Keyes, Paul D. Hovland, Lois C. McInnes, and Widodo Samyono, *Using automatic differentiation for second-order matrix-free methods in PDE-constrained optimization*, in Corliss et al. [42], pp. 35 – 50.

[99] Andreas Kirsch, *An introduction to the mathematical theory of inverse problems*, Springer Series in Applied Mathematical Sciences, vol. 120, Springer, Berlin, 1996.

[100] Arnulf Kost, *Numerische Methoden in der Berechnung elektromagnetischer Felder*, Springer, Berlin - Heidelberg - New York, 1995.

[101] Michael Kuhn, Ulrich Langer, and Joachim Schöberl, *Scientific computing tools for 3D magnetic field problems*, The Mathematics of Finite Elements and Applications (J. R. Whiteman, ed.), vol. X, Elsevier, Amsterdam, 2000, pp. 239 – 258.

[102] Michael Kuhn, Dalibor Lukáš, and Wolfram Mühlhuber, *An object-oriented library for shape optimization problems governed by systems of linear elliptic partial differential equations*, Trans. VŠB – Techn. Univ. Ostrava, Comput. Sci. Math. Ser. **1** (2001), no. 1, 115 – 128.

[103] Karl Kunisch and Gunther Peichl, *Embedding domain technique representation of the gradient for some shape optimization problems*, Adv. Math. Sci. Appl. **9** (1999), no. 2, 717 – 736.

[104] Ulrich Langer and Werner Queck, *On the convergence factor of Uzawa's algorithm*, J. Comput. Appl. Math. **15** (1986), 191 – 202.

[105] ______, *Preconditioned Uzawa-type iterative methods for solving mixed finite element equations. theory – applications – software*, Wissenschaftliche Schriftenreihe 3, Techn. Univ. Karl-Marx-Stadt, 1987.

[106] Reinhard Lerch, Manfred Kaltenbacher, Hermann Landes, and Franz Lindinger, *Computerunterstützte Entwicklung elektromechanischer Transducer*, e & i **7 / 8** (1996), 532 – 545.

[107] David G. Luenberger, *Introduction to linear and nonlinear programming*, Addison-Wesley Publishing Company, 1973.

[108] Dalibor Lukáš, *Shape optimization of homogeneous electromagnets*, SFB-Report 00-30, SFB F013, University of Linz, Austria, September 2000.

[109] ______, *Shape optimization of homogeneous electromagnets*, Proceedings of SCEE, Lecture Notes in Computational Science and Engineering, Springer, 2000, submitted.

[110] Bernd Maar and Volker Schulz, *Interior point multigrid methods for topology optimization*, Struct. Multidisc. Optim. **19** (2000), no. 3, 214 – 224.

[111] Kamel G. Mahmoud, *Approximations in optimum structural design*, Advances in Structural Optimization (Barry H. V. Topping and Manolis Papadrakakis, eds.), Civil-Comp Press, Edingburgh, 1994, pp. 57 – 67.

[112] Gérard Meurant, *Computer solution of large linear systems*, Studies in Mathematics and its Applications, vol. 28, Elsevier, Amsterdam, 1999.

[113] Bijan Mohammadi and Olivier Pironneau, *Applied shape optimization for fluids*, Numerical Mathematics and Scientific Computation, Oxford Science Publications, Clarendon Press, Oxford, 2001.

[114] Frithiof Niordson, *Optimal design of elastic plates with a constraint on the slope of the thickness function*, Internat. J. Solids Structures **19** (1983), 141 – 151.

[115] Jorge Nocedal and Stephen J. Wright, *Numerical optimization*, Springer Series in Operations Research, Springer, New York, 1999.

[116] Sigeru Omatu and John H. Seinfeld, *Distributed parameter systems. theory and applications*, Oxford Mathematical Monographs, Clarendon Press, Oxford, 1989.

[117] Christopher C. Paige and Michael A. Saunders, *Solution of sparse indefinite linear systems of linear equations*, SIAM J. Numer. Anal. **12** (1975), 617 – 629.

[118] Joakim Petersson, *On stiffness maximization of variable thickness sheet with unilateral contact*, Quart. Appl. Math. **54** (1996), 541 – 550.

[119] ———, *A finite element analysis of optimal variable thickness sheets*, SIAM J. Numer. Anal. **36** (1999), no. 6, 1759 – 1778.

[120] ———, *Some convergence results in perimeter-controlled topology optimization*, Comput. Methods Appl. Mech. Engrg. **171** (1999), 123 – 140.

[121] Joakim Petersson and Ole Sigmund, *Slope constrained topology optimization*, Internat. J. Numer. Methods Engrg. (1998), 1417 – 1434.

[122] Michael J. D. Powell, *A fast algorithm for nonlinearly constraint optimization calculations*, Numerical Analysis (G. A. Watson, ed.), Lecture Notes in Mathematics, vol. 630, Springer, Berlin, 1978, pp. 144 – 157.

[123] Alfio Quarteroni and Alberto Valli, *Numerical approximation of partial differential equations*, Springer Series in Computational Mathematics, vol. 23, Springer, Berlin - Heidelberg - New York, 1994.

[124] Werner Queck, *The convergence factor of preconditioned algorithms of the Arrow-Hurwicz type*, SIAM J. Numer. Anal. **26** (1989), no. 4, 1016 – 1030.

[125] Darren Redfern, *The MAPLE handbook – Maple V Release 4*, Springer, 1996.

[126] Stefan Reitzinger, *Algebraic multigrid methods for large scale finite element equations*, Schriften der Johannes-Kepler-Universität Linz, Reihe C, vol. 36, Universitätsverlag Rudolf Trauner, Linz, 2001.

[127] Michael Renardy and Robert C. Rogers, *An introduction to partial differential equations*, Texts in Applied Mathematics, vol. 13, Springer, New York, 1992.

[128] Mark P. Rossow and John E. Taylor, *A finite element method for the optimal design of variable thickness sheets*, AIAA J. **11** (1973), 1566 – 1569.

[129] George I. N. Rozvany, *Aims, scope, methods, history and unified terminology of computer-aided topology optimization in structural mechanics*, Struct. Multidisc. Optim. **21** (2001), no. 2, 90 – 108.

[130] John W. Ruge and Klaus Stüben, *Algebraic multigrid (AMG)*, Multigrid Methods, Frontiers in Applied Mathematics, vol. 3, SIAM, 1986, pp. 73 – 130.

[131] Yousef Saad, *Sparskit: A basic tool-kit for sparse matrix computations*, Tech. Report 90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990, for version 2 and additional information see also `http://www.cs.umn.edu/Research/arpa/SPARSKIT/sparskit.html`.

[132] ———, *Iterative methods for sparse linear systems*, PWS, 1996, second edition available under `http://www.cs.emn.edu/~saad/books.html`.

[133] Yousef Saad and Martin H. Schultz, *GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems*, SIAM J. Sci. Statist. Comput. **7** (1986), 856 – 869.

[134] Ekkehard W. Sachs, *Control applications of reduced SQP methods*, Computational Optimal Control. Proceedings of the 9th IFAC Workshop on Control Applications of Optimization (R. Bulirsch and D. Kraft, eds.), Birkhäuser, 1994, pp. 89 – 104.

[135] Alexander A. Samarskij, *Theorie der Differenzenverfahren*, Mathematik und ihre Anwendung in Physik und Technik, vol. 40, Akademische Verlagsgesellschaft Geest & Porting K.-G., Leipzig, 1984.

[136] Klaus Schittkowski, *On the convergence of a sequential quadratic programming method with an augmented Lagrangian search direction*, Mathematische Operationsforschung und Statistik **14** (1983), 197 – 216.

[137] ———, *NLPQL: A Fortran subroutine for nonlinear programming*, Ann. Oper. Res. **5** (1985), 485 – 500.

[138] Volker Schulz, *Solving discretized optimization problems by partially reduced SQP methods*, Comput. Visualization Sci. **1** (1998), 83 – 96.

[139] Volker Schulz and Hans-Georg Bock, *Partially reduced SQP methods for large-scale nonlinear optimization problems*, Nonlinear Anal. (1997), 4723 – 4734.

[140] Ole Sigmund, *Design of material structures using topology optimization*, Ph.D. thesis, DCAMM, Technical University of Denmark, 1994.

[141] Jan Sokolowski and Jean-Paul Zolésio, *Introduction to shape optimization*, Springer Series in Computational Mathematics, vol. 16, Springer, Berlin - Heidelberg - New York, 1992.

[142] Christoph Stangl, *Optimal sizing for a class of nonlinearly elastic materials*, SIAM J. Optim. **9** (1999), no. 2, 414 – 443.

[143] Josef Stoer, *Numerische Mathematik 1*, 8-th ed., Springer, Berlin, 1999.

[144] Mathias Stolpe and Krister Svanberg, *An alternative interpolation scheme for minimum compliance topology optimization*, Report TRITA/MAT-00-OS13, Optimization and Systems Theory, Royal Institute of Technology, Stockholm, Sweden, 2000.

[145] Krister Svanberg, *The method of moving asymptotes - a new method for structural optimization*, Internat. J. Numer. Methods Engrg. **24** (1987), 359 – 373.

[146] Shlomo Ta'asan, *Infinite dimensional preconditioners for optimal design problems*, Tech. report, Carnegie Mellon University, Pittsburgh, PA, 1997.

[147] ———, *Introduction to shape design and control*, Tech. report, Carnegie Mellon University, Pittsburgh, PA, 1997.

[148] ———, *Multigrid one-shot methods and design strategy*, Tech. report, Carnegie Mellon University, Pittsburgh, PA, 1997.

[149] ———, *Theoretical tools for problem setup*, Tech. report, Carnegie Mellon University, Pittsburgh, PA, 1997.

[150] TAF, see `http://www.fastopt.de/taf/taf.html`.

[151] Ulrich Tautenhahn and Dietmar Schweigert, *Effective numerical methods for solving implicit ill-posed inverse problems*, Theory and Practice of Geophysical Data Inversion (Andreas Vogel, ed.), Vieweg, Braunschweig - Wiesbaden, 1992, pp. 3 – 19.

[152] Ulrich Trottenberg, Cornelis W. Oosterlee, and Anton Schüller, *Multigrid*, Academic Press, London, 2000.

[153] Ursula van Rienen, *Numerical methods in computational electrodynamics*, Lecture Notes in Computational Science and Engineering, vol. 12, Springer, Berlin - Heidelberg - New York, 2000.

[154] Rüdiger Verfürth, *A review of a-posteriori error estimation and adaptive mesh-refinement techniques*, Wiley-Teubner Series on Advances in Numerical Mathematics, John Wiley & Sons, Chinchester, 1996.

[155] Gerhard Wachutka, *Tailored modeling of miniaturized electrothermomechanical systems using thermodynamic methods*, Micromechanical Systems **40** (1992), 183 – 198.

[156] Thomas Weiland, *A discretization method for the solution of Maxwell's equation for six-component fields*, Electronics and Communications AEÜ **3** (1977), 116 – 120.

[157] Wolfgang L. Wendland, *On asymptotic error estimates for combined BEM and FEM*, Finite Element and Boundary Element Techniques from Mathematical and Engineering Point of View (Erwin Stein and Wolfgang L. Wendland, eds.), CISM Cources and Lectures, vol. 301, Springer, Vienna - New York, 1989, pp. 273 – 333.

[158] Stephen Wolfram, *The MATHEMATICA book*, 4th ed., Cambridge University Press, Cambridge, 1999.

[159] Mihalis Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Algebraic Discrete Methods **2** (1981), 77 – 79.

[160] Eberhard Zeidler, *Nonlinear functional analysis and its application*, vol. III, Variational Methods and Optimization, Springer, New York, 1985.

[161] M. Zhou, Y. K. Shyy, and H. L. Thomas, *Checkerboard and minimum member size control in topology optimization*, WCSMO-3, Proceedings of the Third World Congress of Structural and Multidisciplinary Optimization (C. L. Bleobaum, K. E. Lewis, and R. W. Mayne, eds.), 1999, pp. 440 – 442.

[162] Franz Ziegler, *Technische Mechanik der festen und flüssigen Körper*, Springer, Wien - New York, 1985.

[163] Olgierd C. Zienkiewics, *The finite element method in engineering science*, third ed., McGraw-Hill, London, 1977.

[164] Walter Zulehner, *Analysis of iterative methods for saddle point problems: A unified approach*, Math. Comp. **71** (2002), 479 – 505.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, daß ich die vorliegende Dissertation selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Linz, im April 2002 Dipl.-Ing. Wolfram Mühlhuber

Curriculum Vitae

Name: Wolfram Friedrich Mühlhuber

Date of Birth: March 7, 1972

Place of Birth: Linz

Nationality: Austria

Affiliation: Special Research Program (SFB) F013
Numerical and Symbolic Scientific Computing
Institute of Computational Mathematics
Johannes Kepler University Linz
Freistädter Straße 313, A-4040 Linz, Austria
`http://www.sfb013.uni-linz.ac.at`

Education:

1982 - 1990:	Grammer School
1990 - 1997:	Studies in Technical Mathematics at the Johannes Kepler University Linz, graduated to Dipl.-Ing. in September 1997
since 1998:	PhD student at the University of Linz

Employment:

Since 1998:	Research Assistant at the SFB F013